Pre-Trained Denoising Autoencoders Long Short-Term Memory Networks as probabilistic Models for Estimation of Distribution Genetic Programming

---

Student: Roman Höhn

Date of Birth: 1991-04-14

Place of Birth: Wiesbaden, Hesse

Student ID: 2712497

Supervisor: David Wittenberg

Master Thesis - M.Sc. Business Education

FB 03: Chair of Business Administration and Computer Science

Johannes Gutenberg University Mainz

Date of Submission: 2023-02-16

Contents

List of Tables

List of Figures

Abstract

English

Denoising Autoencoder Genetic Programming (DAE-GP) is an Estimation of Distribution Algorithm in the domain of Genetic Programming that uses Denoising Autoencoders Long Short-Term Memory Networks (DAE-LSTM) as probabilistic models for sampling new populations of solutions. This thesis investigates the possible benefits and downsides of using pre-training for the DAE-LSTM networks of DAE-GP for four real world symbolic regression problems. The experiments conducted did show that pre-training can drastically reduce the number of epochs that are necessary for the DAE-LSTM training at each generation of the DAE-GP search. Another interesting finding was that pre-training also increases the levenshtein edit distance between individual solutions inside the population which is a metric for the diversity of a population. Unfortunately pre-training did not show any improvements for both final fitness and the final size of solutions while largely increasing the total run-time for DAE-GP.

Deutsch

Denoising Autoencoder Genetic Programming (DAE-GP) ist ein Estimation of Distribution Algorithmus (EDA) aus dem Forschungfeld der genetischen Programmierung (GP). In DAE-GP werden Denoising Autoencoders Long Short-Term Memory Netzwerke (DAE-LSTM) als probabilistische Modelle verwendet um neue Populationen von Lösungen für eine evolutionäre Suche zu erzeugen. Diese Masterarbeit untersucht die Vor- und Nachteile des Einsatzes einer Pre-Training Strategie für die DAE-LSTM Netzwerke von DAE-GP an vier Datensätzen für symbolische Regression. Die durchgeführten Experimente haben gezeigt, dass Pre-Training die Anzahl von Trainingsepochen für die DAE-LSTM Netzwerke in jeder Generation statistisch signifikant reduzieren können. Außerdem zeigt sich, dass Pre-Training die Levenshtein Editierdistanz, ein Maß für die Populationsdiversität, signifikant erhöhen konnte. Leider konnte Pre-Training die Qualität der jeweils besten gefundenen Lösungen, weder im Bezug auf ihre Fitness noch auf ihre Größe, verbessern. Auch führte Pre-Training in den durchgeführten Experimenten zu einer starken Erhöhung der Laufzeit des DAE-GP Algorithmus.

Keywords

Genetic Programming, Estimation of Distribution Algorithms, Denoising Autoencoder Genetic Programming, Pre-Training, Long Short-Term Memory Networks, Symbolic Regression

1 Introduction

Denoising Autoencoder Genetic Programming (DAE-GP) is a novel variation of an genetic programming based Estimation of Distribution Algorithm (EDA-GP) that uses a denoising autoencoders long short-term memory network (DAE-LSTMs) as a probabilistic model to sample new candidate solutions [35].

DAE-LSTMs are artificial neural networks that can be trained in an unsupervised learning environment to minimize a reconstruction error for encoding input data into a compressed representation and subsequently decoding the compressed representation back to the input dimension. In DAE-GP, DAE-LSTMs are trained with a subset of high-fitness solutions selected from a parent population with the aim to capture their promising qualities. The resulting model is then used to sample new offspring solutions by propagating partially mutated solutions from the parent population through the DAE-LSTM [35]. In previous work DAE-GP has been shown to outperform GP for both a generalized version of the royal tree problem [35] as well as for a real-world symbolic regression problem [34].

The DAE-GP algorithm first described by [35] trains a DAE-LSTMs for each generation g of the search from scratch. [34] and [33] suggest the incorporation of a pre-training strategy into the evolutionary search as a possible way of improving the overall performance of the DAE-GP algorithm. The key idea is to pre-train an initial DAE-LSTM on a large population of candidate solutions and to use the pre-trained parameters of this initial model in each generation of the search as starting parameters for the current generation DAE-LSTM. This thesis studies the influence of using pre-trained DAE-LSTMs in DAE-GP for symbolic regression, especially looking at the influence on overall quality of solutions found by DAE-GP and effects on run-time. The aim of this study is to answer the question if a pre-training strategy can be used to improve DAE-GP and to study both positive and negative effects on overall performance.

2 Theoretical Foundations

This section describes the relevant concepts that are necessary for the understanding and classification of DAE-GP as well as the concept of pre-training in artificial neural networks.

2.1 Denoising Autoencoder Genetic Programming

DAE-GP is an EDA-GP algorithm that uses DAE-LSTM networks as a probabilistic model to sample new offspring solutions [35].

2.1.1 Evolutionary Computation

As a variant of GP, DAE-GP can be classified as a meta-heuristic that belongs to the field of evolutionary computation (EC). EC based meta-heuristics are optimization methods that simulate the process of Darwinian evolution to search for high quality solutions by applying selection and variation operators to a population of candidate solutions. Examples of EC include genetic algorithms (GA), evolutionary strategies (ES) and GP. In EC, the quality

of a solution is commonly measured as fitness and the time steps of the search are called generations. Another important concept in EC is the distinction between genotypes and phenotypes of solutions. The genotype contains the information that is necessary to construct the phenotype, the outer appearance of a particular solution on which we measure the overall quality of solutions. The representation of a solution is therefore defined by the mapping of genotypes to phenotypes [22]. Genetic operators, such as mutation or recombination, are usually applied to the genotype of solutions.

2.1.2 Genetic Programming

GP follows the same basic evolutionary principle of EC but searches for more general, hierarchical computer programs of dynamically varying size and shape [15]. The computer programs that are at the center of the evolutionary search in GP are commonly represented by tree structures at the level of their phenotype [22]. Since GP searches for high fitness computer programs that produce a desired output for some input, it can be applied to various different problem domains such as symbolic regression, automatic programming, or evolving game-playing strategies [15]. An important quality of GP is the ability to search for solutions of variable length and structure. GP is an especially useful meta-heuristic for problems where no a priori knowledge about the final form of good solutions is available. GPs ability to optimize solutions for both their structure as well as for their parameters led to it being one of the most prevalent methods used for symbolic regression [20].

2.1.3 Estimation of Distribution Algorithms

The aim of estimation of distribution algorithms (EDA) is to replace the standard variation operators used in EC by building probabilistic models that can capture complex dependencies between different decision variables of an optimization problem [22]. EDAs use this probabilistic model to sample new offspring solutions inside an evolutionary search to replace crossover and/or mutation operators. Even though a majority of EDA algorithms are designed to work with fixed length string representations as used in classical GAs, EDA algorithms have been researched and successfully applied to the domain of GP in various different research streams [13].

2.1.4 Denoising Autoencoders

One possible way of model building in EDA proposed by [21] is to use denoising autoencoders (DAE) as generative models to capture complex probability distributions of decision variables. DAE, a variation of the autoencoder (AE), is a widely used type of neural networks in the domain of unsupervised machine learning that maps n input variables to n output variables using a hidden representation.

AE were introduced by [10] to compress high-dimensional data into lower-dimensions. An AE consists of two different sub-units:

- Encoder $g(x)$: Encode input data to a smaller central layer h
- Decoder $d(h)$: Decode and output the encoded data back to its original dimension

The AE is trained to reduce the reconstruction error between input and output data. After the training procedure is finished the network is able to reduce the dimensionality of input data to a compressed representation[10].

DAE was first introduced by [29] as an improved AE with the ability to learn new representations of data that is especially robust to partially corrupted input data. DAE modifies the AE by using partially corrupted input data for the AE and training it to reconstruct the uncorrupted, original version of the input data.

Since the hidden representation of DAE captures the dependency structure of the input data it can therefore also be used to generate new solutions in the context of GAs [21].

DAE-GP builds upon the concept of using DAEs in EDAs described by [21] and transfers the concept to the domain of GP. The mutation and crossover operators of standard GP are replaced by sampling new solutions from a probabilistic model that is build by training a DAEs long short-term memory (LSTM) network on a subset of high fitness solutions from the current population [35].

LSTMs are a variant of artificial neural networks first introduced by [11] that can store learned information over en extended period of time while avoiding the problem of vanishing and/or exploding gradients. Since DAE-GP encodes candidate solutions as linear strings in prefix notation, the DAE in DAE-GP uses LSTMs for both encoding and decoding where the total amount of time steps T is equal to the sum of the length of the input solution and the output solution [35].

2.2 Pre-Training

Pre-Training describes the concept of initially training an artificial neural network on a large dataset before using it to solve a more specific task. It is a commonly used strategy in deep architectures that has been shown to improve both the optimization process itself as well as the generalization behavior if compared to the standard approach of using randomly initialized parameters [5].

The strategy of pre-training artificial neural networks is based on the idea of transfer learning that aims at retaining previously learned knowledge from one task to re-use it for another task [7] [19]. Transfer learning traditionally follows a two phase approach:

1. Pre-Training Phase: Capture knowledge from source task
2. Fine-Tuning Phase: Transfer knowledge to the target task

where source task and target task are usually similar but may differ in their feature space and the distribution of training data [19].

One of the main benefits of using pre-training is the ability to reduce the need for large amounts of training data which can often be unavailable or too expensive to collect. Additionally, pre-training can lead to an improved performance by either reducing the computational effort that is necessary to train a model to solve a specific task or by improving the models generalization ability.

3 Pre-Training in DAE-GP

The pre-training strategy used in this thesis is applied to the DAE-LSTM models M_g that are used in each DAE-GP generation g for $g \in \{x \mid x \text{ is a number and } 1 \leq x \leq g_{max}\}$. Instead of initializing and optimizing each model from scratch as done in previous work (e.g. [35] [34]), a separate DAE-LSTM network $\hat{M}$ will be trained on a large population of randomly initialized solutions.

The trainable parameters $\theta_{\hat{M}}$ obtained after finishing the training procedure of $\hat{M}$ are then used as the starting parameters for each following DAE-LSTM model M_g.

The motivation for incorporating pre-training into DAE-GP is based on the following suspected mechanisms of improvement:

1. Improve overall performance of DAE-GP by improving either run time and/or solution quality
2. Reduce the need for large population sizes to avoid sampling error
3. Improve model robustness and generalization ability

The probably most obvious motivation for using a pre-training strategy in DAE-GP is based on the fact that initializing M_g with pre-trained weights $\theta_{\hat{M}}$ is likely to reduce the amount of training epochs that are necessary until the point of training error convergence is reached.

Early research on pre-training for DAE by [5] comes to the conclusion that besides adding robustness to models the strategy also results in improved generalization and better performing models. The authors describe that unsupervised pre-training behaves like a regularizer on DAE networks. The mean test error and its variance are reduced for DAE networks that are initialized from pre-trained weights if compared to the same architectures that use random initialization.

Another important finding by [5] is that the positive effect of pre-training is dependent on both the depth of the network as well as the size of layers - while increasingly larger networks benefit increasingly more from pre-training the final performance of small architectures tends to be worse with pre-training than with randomized weight initialization. The authors find evidence that pre-training DAE is especially useful for optimizing the parameters of the lower layers of the network.

Another motivation for introducing pre-training into DAE-GP is the prevalence of sampling error for small GP populations sizes. [23] finds that sampling error, non-systematic errors that are caused by observing only a small subset of the statistical population, is a severe problem in the domain of GP since the initial population may not be a representative sample of all possible solutions. [23] also introduces a method for calculating optimal population sizes to minimize the presence of sampling error. Since DAE-LSTM of DAE-GP learns the properties of its training population and reuses the acquired knowledge in the sampling procedure, using the parameters obtained from training $\hat{M}$ on a sufficiently large training population might reduce the need for large population sizes in the following generations of the search if $\hat{M}$ already implicitly captured the properties of a representative sample of

solutions. If this mechanism can be applied successfully it would benefit the performance of DAE-GP by increasing the population diversity (resulting in better solution quality) as well as by reducing the need for large population sizes which require computational resources.

3.1 Implementation

The DAE-GP algorithm, first described by [35], is summarized as a flowchart in figure 1. After setting the generation counter g to 0 and creating the initial population of solutions P_0, DAE-GP enters the main loop that checks if a termination criteria is satisfied (i.e. maximum number of generations or fitness evaluations is reached). For each generation g the fitness of all new individuals in P_g is evaluated. During model building, DAE-GP selects a subset X_g of the current population that is used as training data for the DAE-LSTM model M_g which is trained to learn the properties of X_g. The trained model M_g is then used for model sampling, partially mutated solutions from P_g are propagated through M_g to create a new population P_{g+1}. This process is repeated until the termination criteria is met and DAE-GP returns the highest fitness solutions.

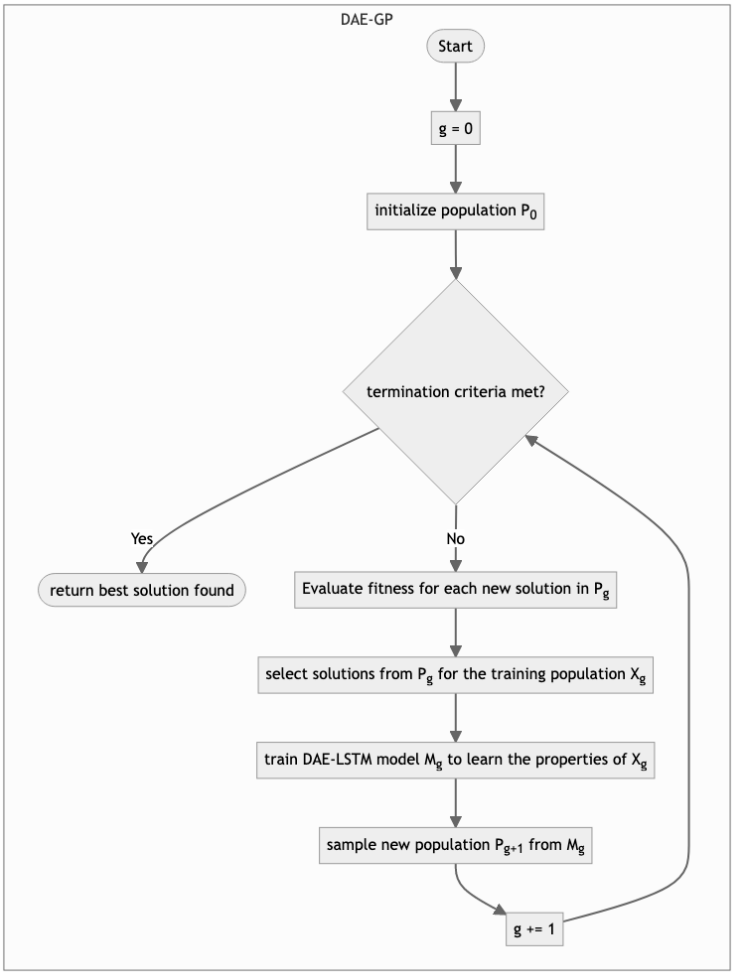

Figure 1: Regular DAE-GP Flowchart

The pre-training strategy implemented for all experiments in this thesis is visualized as another flowchart in figure 2. The main difference to regular DAE-GP is the inclusion of an initial pre-training phase where a separate population $\hat{P}$ is first initialized and then randomly split by half into $\hat{P}_{train}$ and $\hat{P}_{test}$. A DAE-LSTM model $\hat{M}$ is then trained to learn the properties of $\hat{P}_{train}$ using an early stopping training mode where we stop training as soon as the validation error for $\hat{P}_{test}$ converges. After the training for $\hat{M}$ is stopped, the current state of $\hat{M}$ is frozen and the optimized trainable parameters $\theta_{\hat{M}}$ are saved before terminating the pre-training phase to start the DAE-GP phase. During the DAE-GP phase, the only difference to the traditional DAE-GP algorithm described in figure 1 is during model building: We initialize M_g with $\theta_{\hat{M}}$ before training it to learn the properties of X_g.

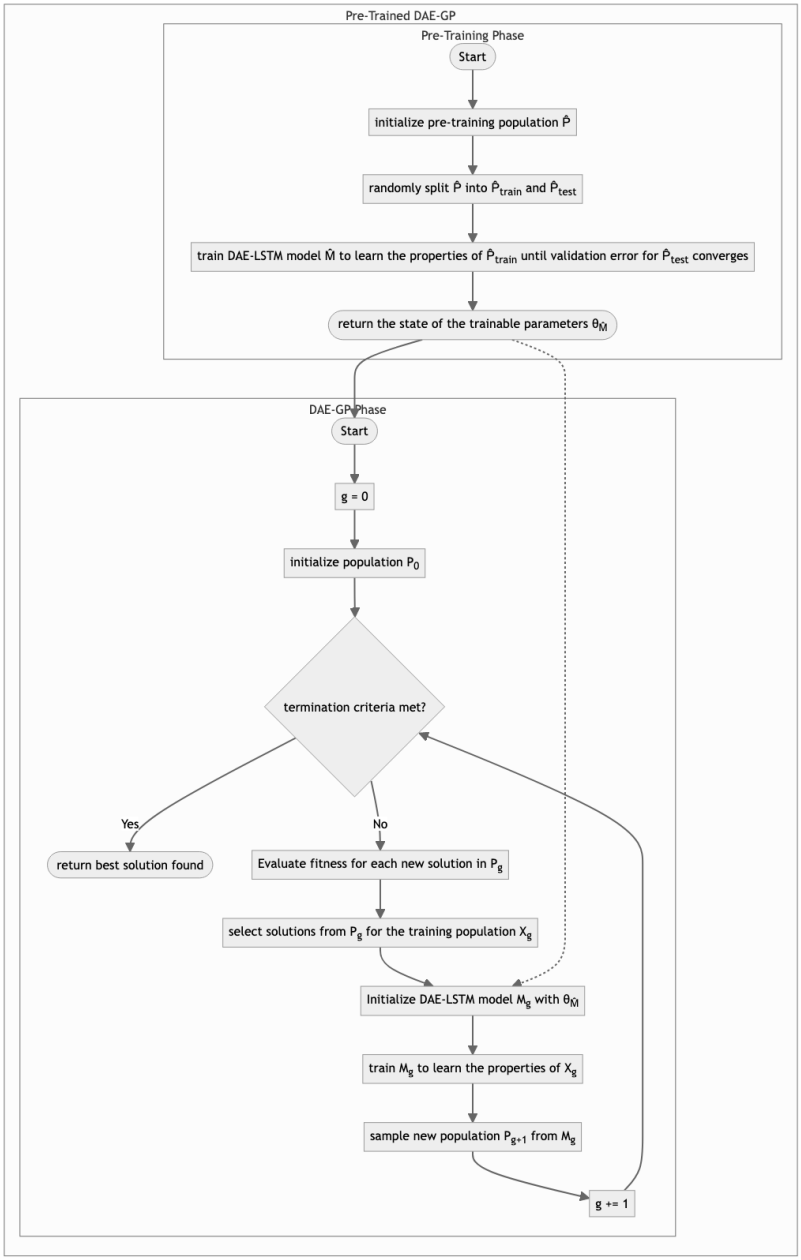

Figure 2: Pre-Trained DAE-GP Flowchart

One difficulty in the implementation of a pre-training strategy into DAE-GP has been the determination of the number of hidden neurons inside the DAE-LSTMs hidden layers. The original description of DAE-GP [35] uses a strategy where the number of hidden neurons for each hidden layer is dynamically set per generation to the maximum individual size inside the current population. For a pre-training implementation, this strategy can not be easily adapted since it leads to a changing number of neurons at each generation resulting in different dimensions of the DAE-LSTM. To allow the sharing of pre-trained parameters $\theta_{\hat{M}}$ from the pre-trained model $\hat{M}$ to each M_g, the number of hidden neurons per hidden layer

for all DAE-GP runs, pre-trained as well as regular, is statically set to a predefined value.

If not stated otherwise, all experiments in this thesis use the GP hyperparameters listed in table 1 and the DAE-LSTM specific hyperparameters listed in table 2.

Table 1: GP - Hyperparameter

Hyperparameter	Value
Population Size	500
Generations	30
Fitness Metric	Root Mean Squared Error (RMSE)
Train/Test Split	50:50
Maximum Tree Depth	17 [15]
Initialization Algorithm	Ramped Half and Half [15]
Min/Max Initialization-Depth	2/6
Selection Operator	Binary Tournament Selection
Function Set	{+ , - , * , AQ [18]}
Ephemeral Constants	[-5,..,5]
Pre-Training Population Size	10000
Pre-Training Train/Test Split	50:50

Table 2: DAE-LSTM - Hyperparameter

Hyperparameter	Value
Stopping Condition	Convergence [33]
Reconstruction Error Metric	Multiclass Cross Entropy
Gradient Descent Algorithm	Adam Optimization [14]
Learning Rate	0.001
Batch Size	0.1
Sampling Steps	1
Hidden Layer	2
Hidden Neurons	150
Corruption Operator	Levenshtein Edit [33]
Denoising Edit Probability (Model Building)	0.05
Denoising Edit Probability (Model Sampling)	0.95
Pre-Training Stopping Condition	Early Stopping

The framework that was used to run all experiments of this thesis was provided by the supervisor of this thesis David Wittenberg and uses the python programming language [27] in conjunction with the baseline libraries `Keras` [2] for deep neural networks and `deap` [6] for evolutionary computation. For data visualizations the python package `matplotlib` [12] was used, statistical analysis and evaluations were conducted using primarily the data science libraries `numpy` [8], `scipy` [30] and `pandas` [25].

3.2 Benchmark Problem

To test pre-training in DAE-GP this thesis focuses on the domain of real-world symbolic regression problems. Symbolic Regression problems have been one of the first GP applications [15] and are an actively studied and highly relevant research area. The goal in symbolic regression is to find a mathematical model for a given set of data points [20]. In real-world symbolic regression these data points are sourced from real-world observations which in contrast to synthetic symbolic regression problem are more likely to contain random noise and bias. An important challenge in solving real-world symbolic regression problems is the ability for a given model to generalize, we want the final model to show high accuracy in predicting outcomes for previously unseen cases.

The main experiments conducted in this thesis uses the NASA Airfoil Self-Noise Data Set [1] that consists of 5 input variables and 1 output variable that are listed in table 3.

Table 3: Airfoil - Dataset Description

Type	**Name**	**Description**	**Unit**
input	x1	Frequency	Hertz
input	x2	Angle of attack	Degree
input	x3	Chord length	meters
input	x4	Free-stream velocity	meters/second
input	x5	Suction side displacement thickness	meters
output	y	Scaled sound pressure level	decibels

The objective of the airfoil problem is to find a function that accurately predicts the output variable y by taking in a subset of the input variables $x1, x2, x3, x4, x5$. The function set used by all DAE-GP variations for symbolic regression is summarized in tables 1. The terminal set consists of the the input variables $x1, x2, x3, x4, x5$ and the ephemeral integer constants listed in table 1 [34].

4 Results for the Airfoil Dataset

4.1 Influence on Generalisation

To gain a deeper understanding about the effect of pre-training the DAE-LSTM networks in DAE-GP, a series of experiments was conducted using the airfoil dataset for symbolic regression while using different parameter configurations for the number of hidden layers as well as the number of hidden neurons per hidden layer.

To study the generalization behavior of the DAE-LSTM model, this series of experiments is conducted using DAE-GP with only a single generation until the search terminates. The fitness of solutions found was disregarded to focus solely on the reconstruction error that is produced during the training of each DAE-LSTM. The reconstruction error is measured for two separate populations, a training population P_{train} that is used to train our DAE-LSTM

as well as a hold-out validation population P_{test}. For the pre-trained DAE-GP two additional, separate populations $\hat{P}_{train}$ and $\hat{P}_{test}$ are used exclusively for pre-training.

DAE-GP is tested in two different configurations:

- Variable number of hidden neurons (50, 100, 200) with a single hidden layer
- Fixed number of hidden neurons (100) per hidden layer with variable number of hidden layers (1, 2, 3)

For each configuration, traditional DAE-GP as well as a pre-trained DAE-GP was tested resulting in 12 total sub experiments that were each based on 10 individual runs (total number of runs=120). To avoid creating biased results through the presence of sampling error inside the pre-training population (see [23]), the population size for the pre-training phase is chosen very high with the size of $\hat{P}$ =10000 where 50% of $\hat{P}$ is used for the training population $\hat{P}_{train}$ and the remaining 50% are used for the hold-out validation population $\hat{P}_{test}$. The training of each DAE-LSTM stopped after a fixed number of 1000 epochs. The reason for using a high amount of 1000 fixed training epochs was to deliberately force the DAE-LSTM to overfit to the training data.

In general, it is expected that with a growing number of trainable parameters (either by adding more hidden layers or more hidden neurons per layer) the DAE-LSTM will be more likely to overfit to the training population P_{train} which will result in a small reconstruction error for the training population and a large one for the validation population P_{test}. The reason for this effect is that a larger network trained over an extended period of time (without the use of strategies like early stopping), has much more potential to learn noise from the training dataset than a smaller network, which is more likely to result in worsening performance for previously unseen cases [31].

Based on the review of [5] it is also expected that pre-training will have an insignificant or even negative influence on small DAE-LSTM instances while improving the networks generalization ability with growing size.

Figure 3 summarizes the median reconstruction error for a variable number of hidden neurons. As expected, increasing the number of hidden neurons leads to stronger overfitting to the training dataset which results in small training errors but growing testing errors. For the experiments run with 50 and 100 hidden neurons, a small benefit of using pre-training can be observed. Using only 50 hidden neurons leads to a smaller gap between training and testing error for the pre-trained DAE-LSTM. For 100 hidden neurons, regular DAE-GP not only starts to overfit to the training data earlier than pre-trained DAE-GP, also it can be seen that pre-trained DAE-GP starts to overfit at a lower reconstruction error which would result in a higher model performance if training strategies such as early stopping would be employed. Doubling the number of hidden neurons to 200 results in very strong overfitting behavior for both DAE-GP variants starting around the 200.th epoch.

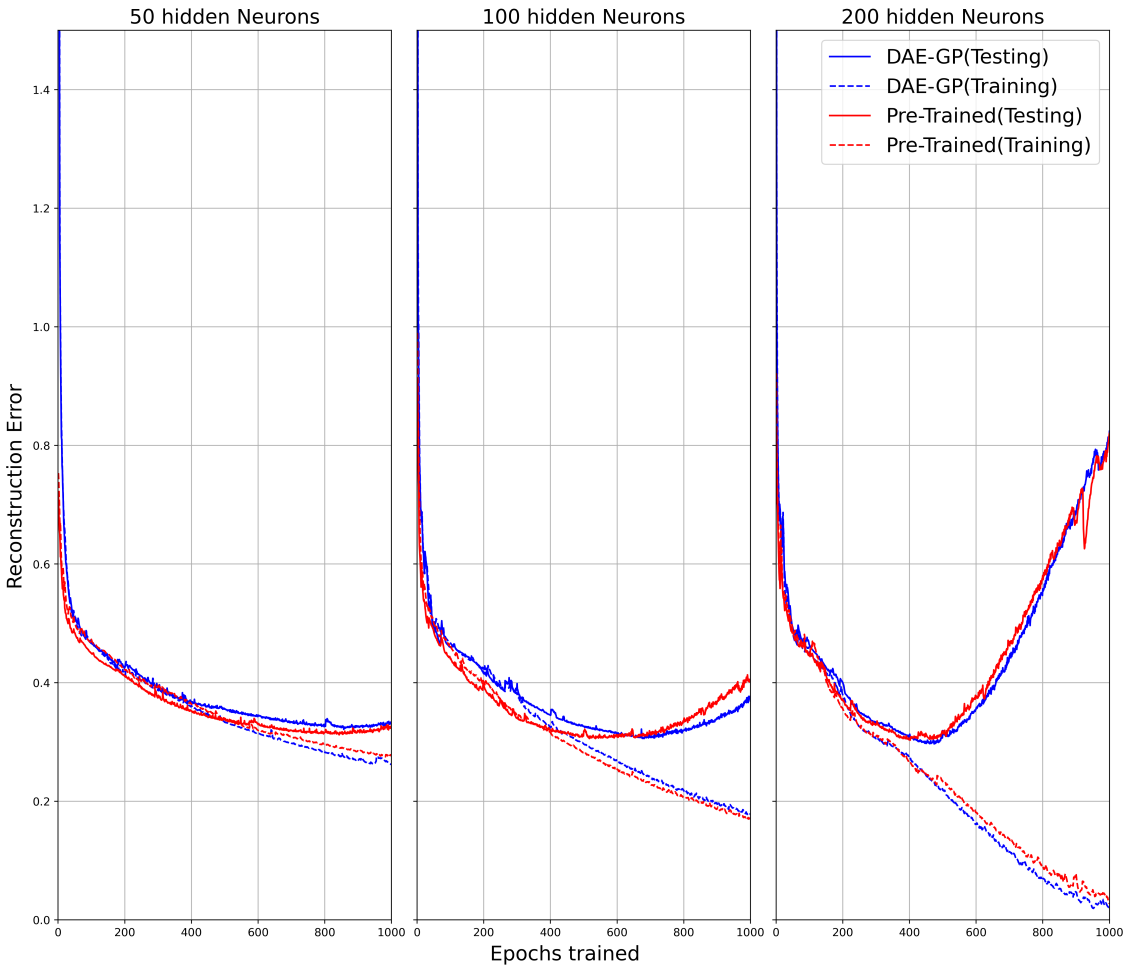

Figure 3: First Generation Median Reconstruction Error for variable number of hidden Neurons

The results for the experiment with a varying number of hidden layers are summarized in figure 4. Again, it can be observed that by increasing the dimension of the DAE-LSTM network both algorithms increasingly overfit to the training data. While for a single hidden layer overfitting begins at around the 400th. epoch, doubling the number of hidden layers to two leads to both algorithms overfitting at around the 200.th epoch. In general over all three sub experiments, both algorithms start to overfit after approximately the same number of epochs but pre-training leads to reaching the point of overfitting at a higher reconstruction error which would result in a inferior model performance if the early stopping training strategy would be used during model training.

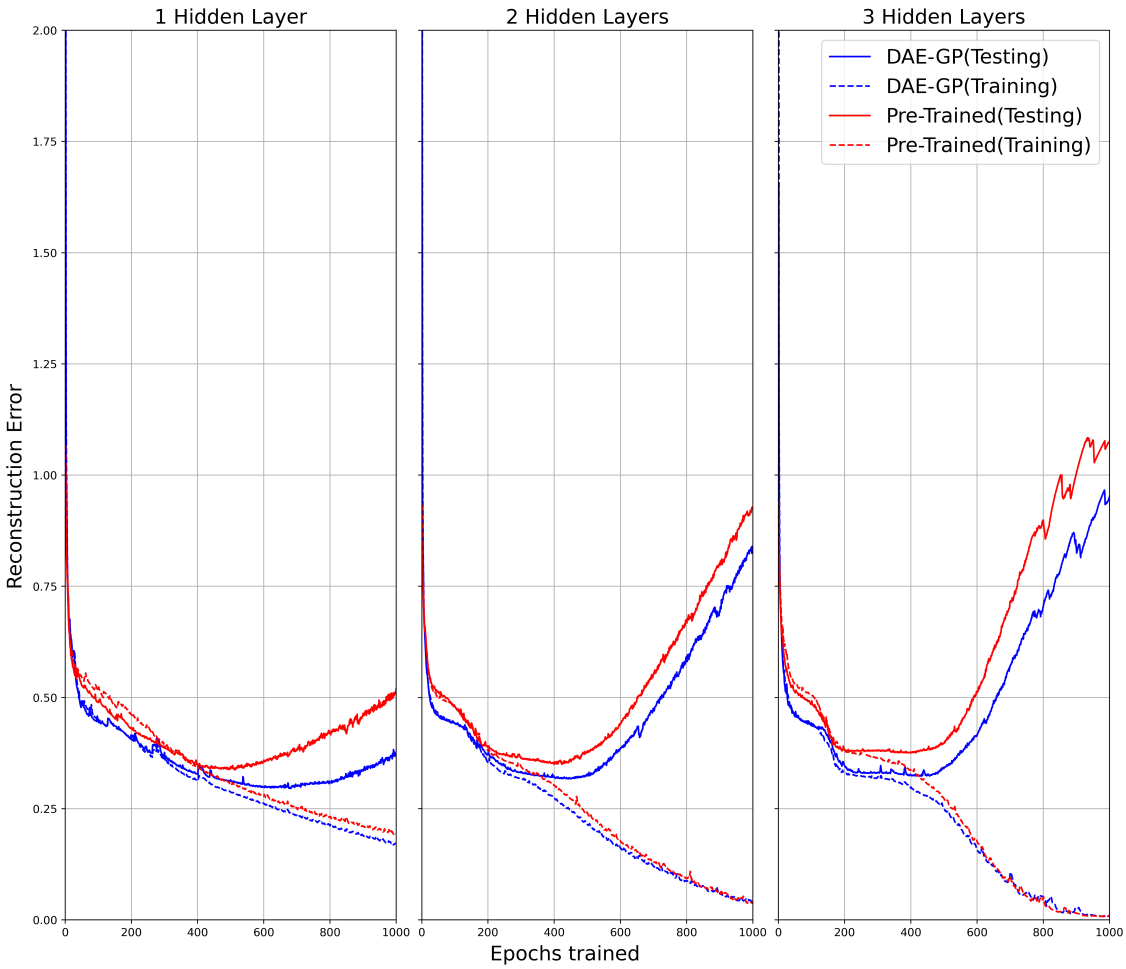

Figure 4: First Generation Median Reconstruction Error for variable number of hidden Layers

4.2 Influence on solution quality

After closely examining the effect of pre-training on DAE-GPs generalization behavior, another series of experiments is conducted to study how pre-training influences the overall search behavior of DAE-GP.

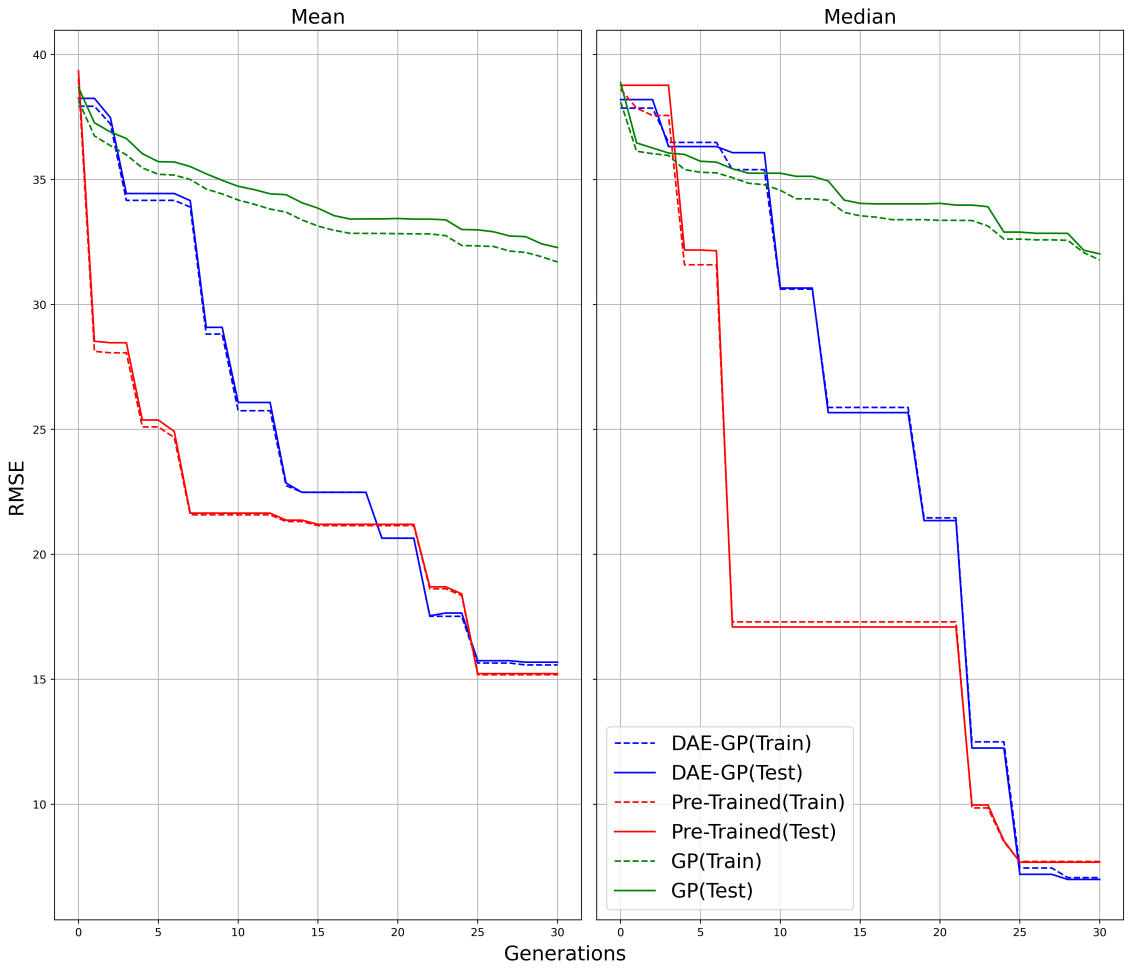

Figure 5: Best Fitness over 30 Generations - Airfoil

Figure 5 shows both the mean and median best fitness by generations for the airfoil dataset. The results are aggregated for 10 individuals runs per algorithm and also include regular GP as a benchmark. For all three algorithms an improvement in the best found fitness can be observed during the evolutionary search but the results show that DAE-GP based algorithms, on average, find solutions with much higher fitness as traditional GP.

The first interesting observation from this experiment is, that the pre-trained DAE-GP shows slightly lower mean RMSE (higher fitness) but a higher median RMSE (lower fitness) than regular DAE-GP for both the training and the testing set in overall fitness. Regarding the presence of overfitting, both DAE-GP variants show only a very small gap between the testing and training fitness which indicates a good generalization behavior.

Another interesting observation from figure 5 is that pre-trained DAE-GP is faster at improving fitness than regular DAE-GP. The median best RMSE on the test set after 10 generations for pre-trained DAE-GP is 17.093 which is smaller by a factor of 0.558 than regular DAE-GP with a median best RMSE of 30.658

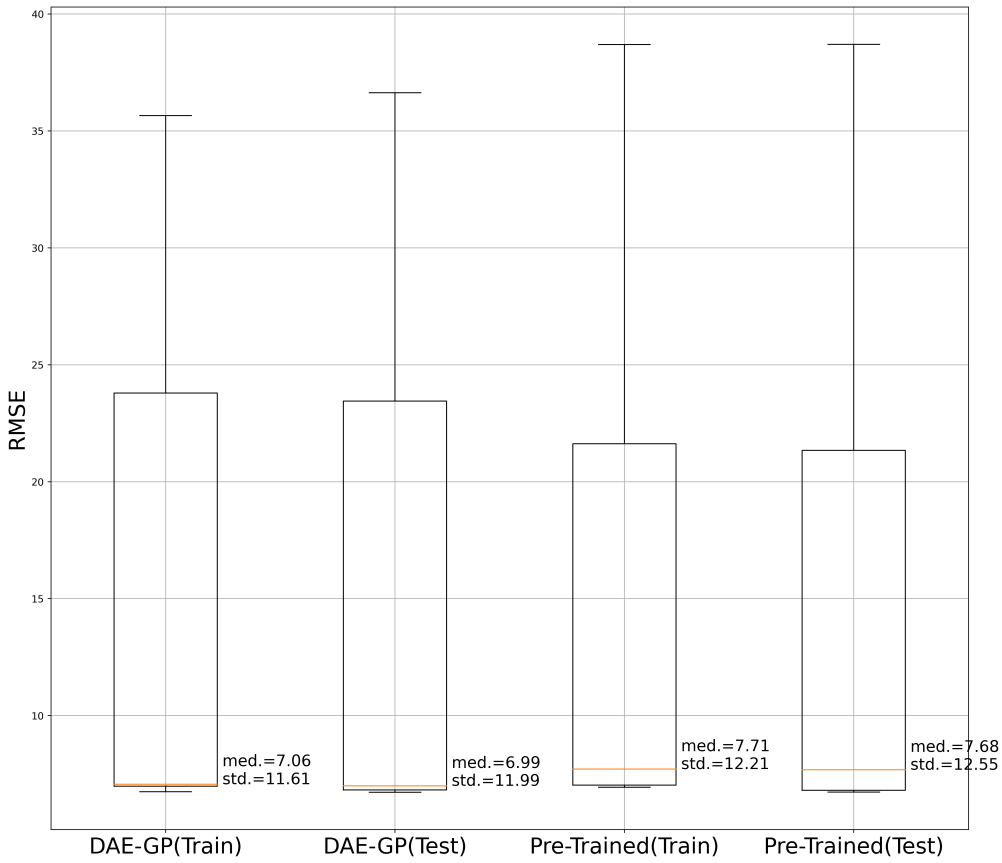

Figure 6: Best Fitness after 30 Generations - Airfoil

The distribution of final fitness scores for pre-trained and regular DAE-GP are visualized as box plots in figure 6. The distribution for regular DAE-GP shows to be preferable to that of pre-trained DAE-GP since it has a lower median RMSE of 6.991 for the test set (7.68 for pre-trained DAE-GP) as well as less dispersion of the solutions measured by a standard deviation of 11.989 (12.548 for pre-trained DAE-GP). For reference, the median final RMSE for the test set of standard GP is at 32.024 which is larger by a factor of 4.17 if compared to pre-trained DAE-GP.

An interesting observation from studying the distribution of final fitness scores for both algorithms is that pre-trained DAE-GP has a smaller interquartile range and more extreme outliers for high RMSE which explains the slightly worse performance if looking at median best fitness present in figure 5.

Since previous work by [34] demonstrated that DAE-GP, for a given number of 10.000 fitness evaluations, is able to produce solutions that are much smaller in tree size than regular GP, the experiment also analyzed the influence of pre-training on the size of solutions found by DAE-GP.

Figure 7 shows the median of both the average size of individuals inside the population as well as the size of the currently best performing solution inside the population. It can be observed that both DAE-GP variants strongly reduce the average solution size already in the first

generations and that the average solution size inside the population constantly stays at this very low level for the rest of the search. The median average size over all generations is 1.536 for DAE-GP and 1.536 for pre-trained DAE-GP. Regarding the size of the best solution per generation, the results show that the pre-trained version of algorithm did produce solutions with the same median size as traditional DAE-GP. The median size of the best solutions found after terminating the search is the same for both algorithms (DAE-GP: 5, Pre-Trained DAE-GP: 5)

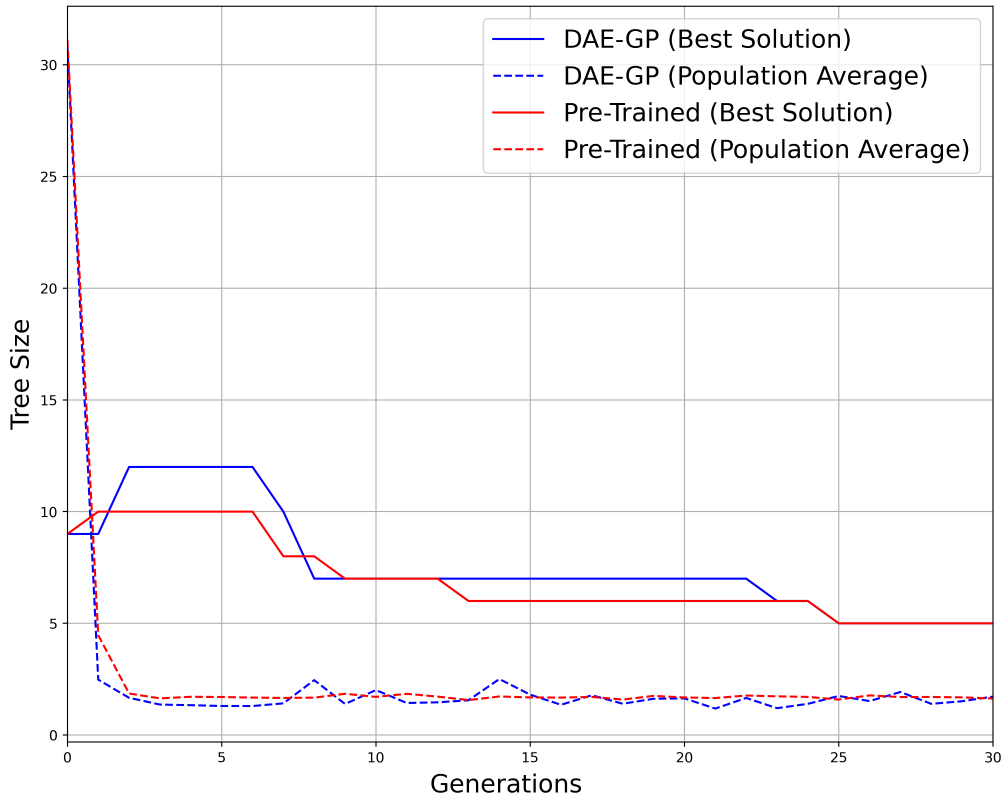

Figure 7: Median Solution Size over 30 Generations - Airfoil

Another interesting metric to examine in population-based optimization algorithms is the population diversity over all generations of the search. Since individual solutions in DAE-GP are computer programs in the form of parse trees, the normalized Levenshtein edit distance [16] was selected as a metric for tracking population diversity. Figure 8 shows the median population diversity for both DAE-GP algorithms and regular GP as a benchmark. While the population diversity steadily decreases over the generations for standard GP, DAE-GP variants strongly increase the population diversity in the first generations and keep them at a higher level during the evolutionary search. Here it can be observed, that pre-training shows a positive effect by increasing population diversity in comparison to regular DAE-GP (Median over all generations for DAE-GP: 0.861, Pre-Trained DAE-GP: 0.925, Standard-GP 0.705).

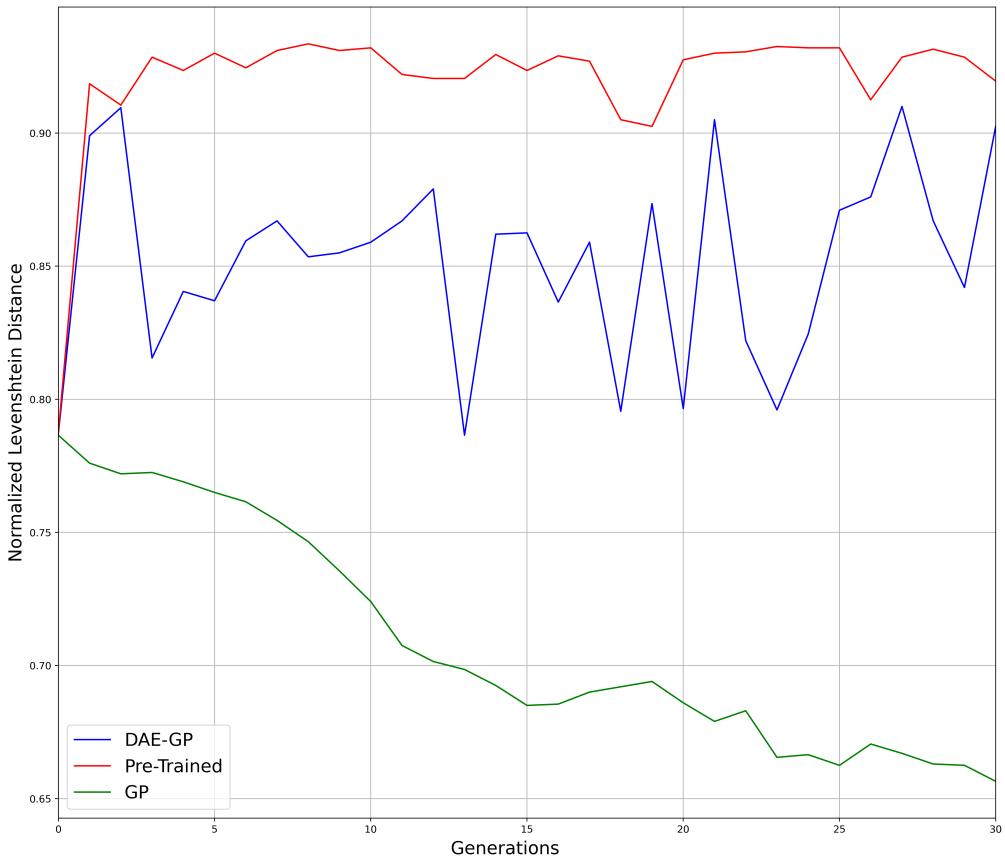

Figure 8: Median Population Diversity over 30 Generations - Airfoil

4.2.1 Dynamic Adjustment of Hidden Neurons

To ensure a fair comparison between pre-trained and regular DAE-GP, the prior experiments used a static number of hidden neurons for each hidden Layer inside the DAE-LSTM networks. For comparison, figure 9 shows the total number of trainable parameters for each generations DAE-LSTM of another experiment where regular DAE-GP was tested with dynamic adjustment of the number of hidden neurons as described in [35] and [34]. Again, the experiment was based on 10 individual runs and otherwise used the same hyperparameters as listed in tables 1 and 2. It clearly shows that DAE-GP can strongly reduce the number of trainable parameters by adjusting the number of hidden neurons per hidden layers to the current generations population which largely reduces the amount of training necessary and likely reduces the run-time of DAE-GP.

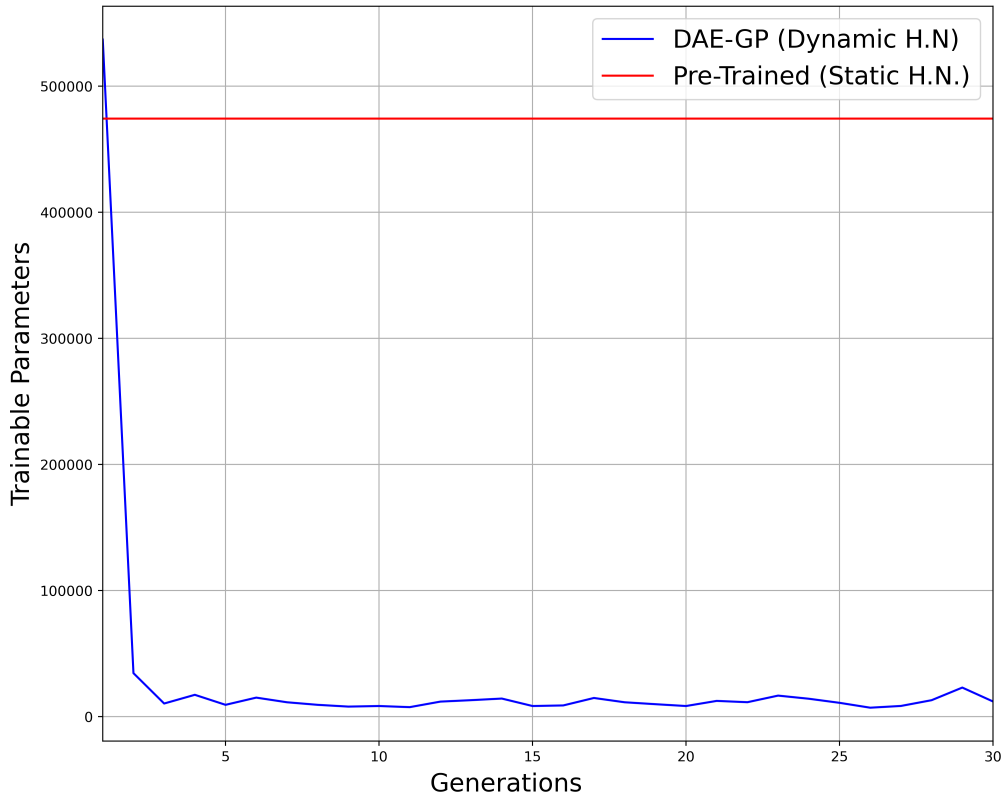

Figure 9: Median Number of trainable Parameters over 30 Generations - Airfoil - Dynamic adjustment of regular DAE-GP

As expected, the reduction of hidden neurons by dynamic adjustment does also result in a reduction of the fitness of solutions found. Figure 10 shows that pre-trained DAE-GP, with its more complex DAE-LSTMs over the full 30 generations, does achieve a higher median fitness than regular DAE-GP with dynamic adjustment of hidden neurons. Pre-Training does therefore introduce a new Trade-Off decision into the adjustment of DAE-GP hyperparameters where the number of hidden neurons needs to be carefully selected since it positively impacts solution quality but also increases the computational cost of DAE-LSTM training.

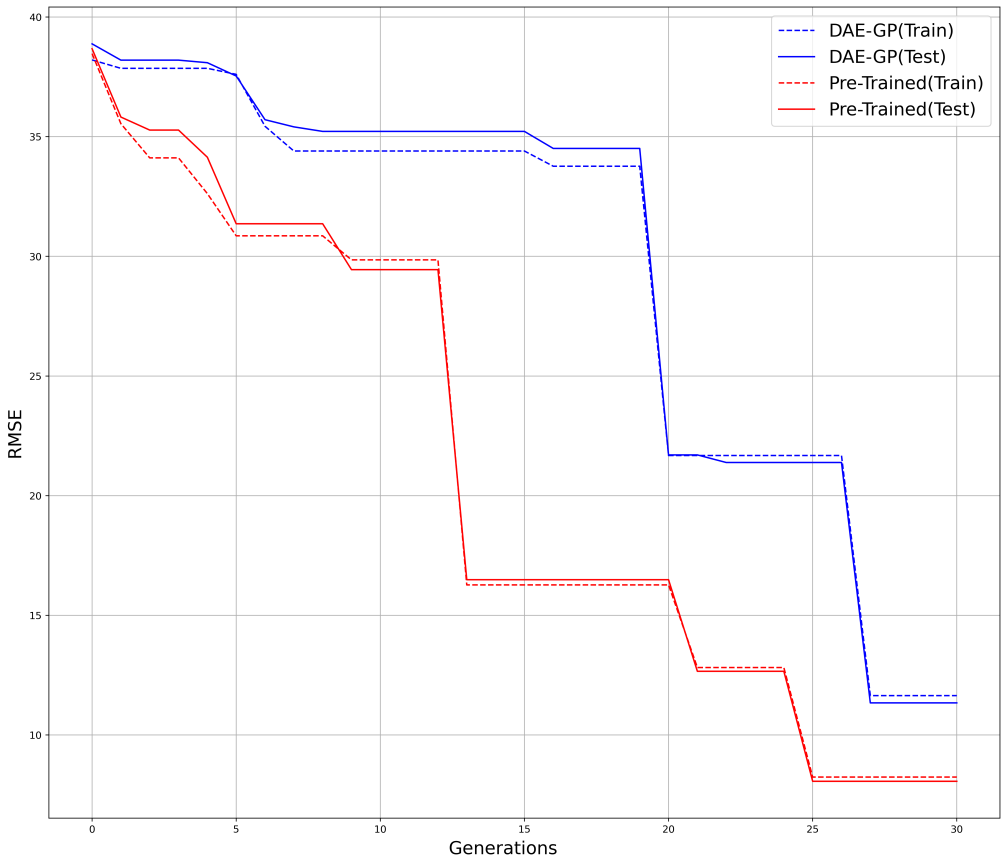

Figure 10: Median Best Fitness over 30 Generations - Airfoil - Dynamic adjustment of regular DAE-GP

4.2.2 Reduced Number of Hidden Layers

Since previous research on DAE-GP used a single hidden layers for the airfoil dataset [34], the experiment was repeated with a single hidden layer. Based on the findings of [5], it is to be expected that pre-training DAE-GP with a single hidden layer inside the DAE-LSTM networks leads to a decrease in overall performance. Figures 11 and 12 show the mean and median best fitness per generation as well as the distribution of final fitness scores after 30 generations. The results are, again, based on 10 individual runs per algorithm. As expected the results show, that for a small DAE-LSTM size pre-training negatively impacts the fitness of solution found by DAE-GP. The median final RMSE for the test set of pre-trained DAE-GP increases by a factor of 1.287 to 16.213 compared to regular DAE-GP with 12.594.

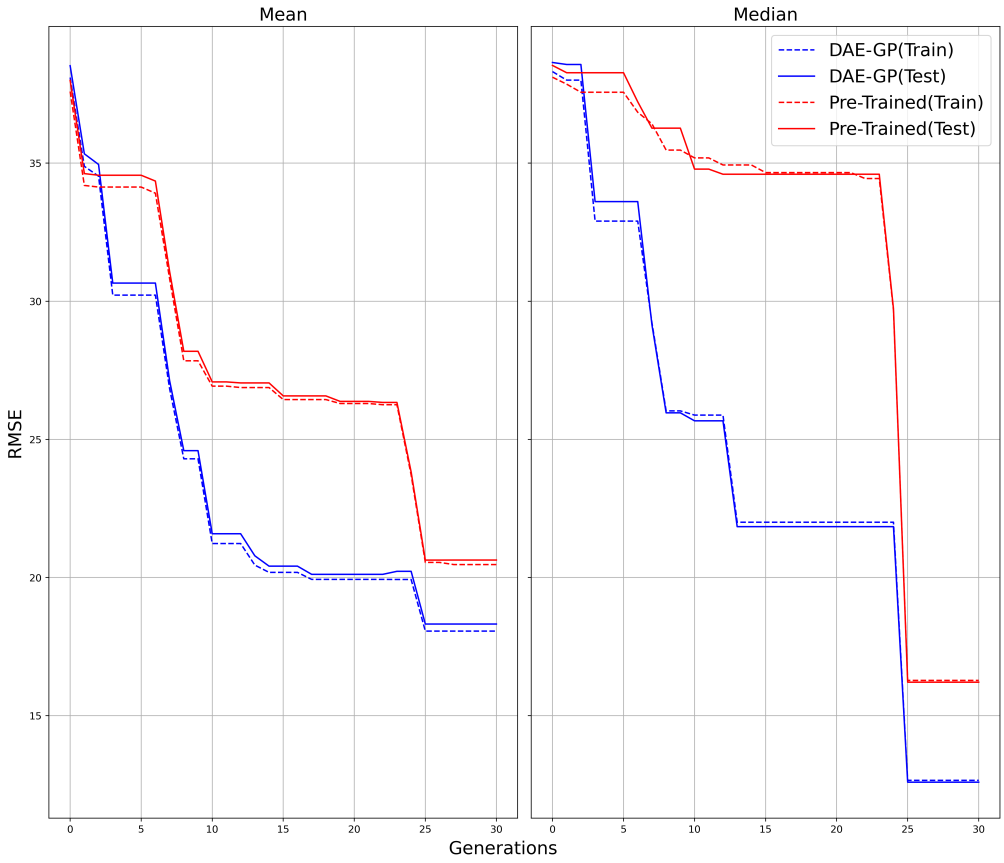

Figure 11: Fitness over 30 Generations - Airfoil - Single Hidden Layer

Interestingly, figure 12 shows that the distribution of final fitness scores for pre-trained DAE-GP has a larger interquartile range than regular DAE-GP. This is opposed to the distributions of final fitness scores that was measured for the same problem run with two hidden layers (see figure 6). Otherwise, pre-trained DAE-GP again has more outliers for high RMSE than regular DAE-GP as seen in the previous experiment with two hidden layers.

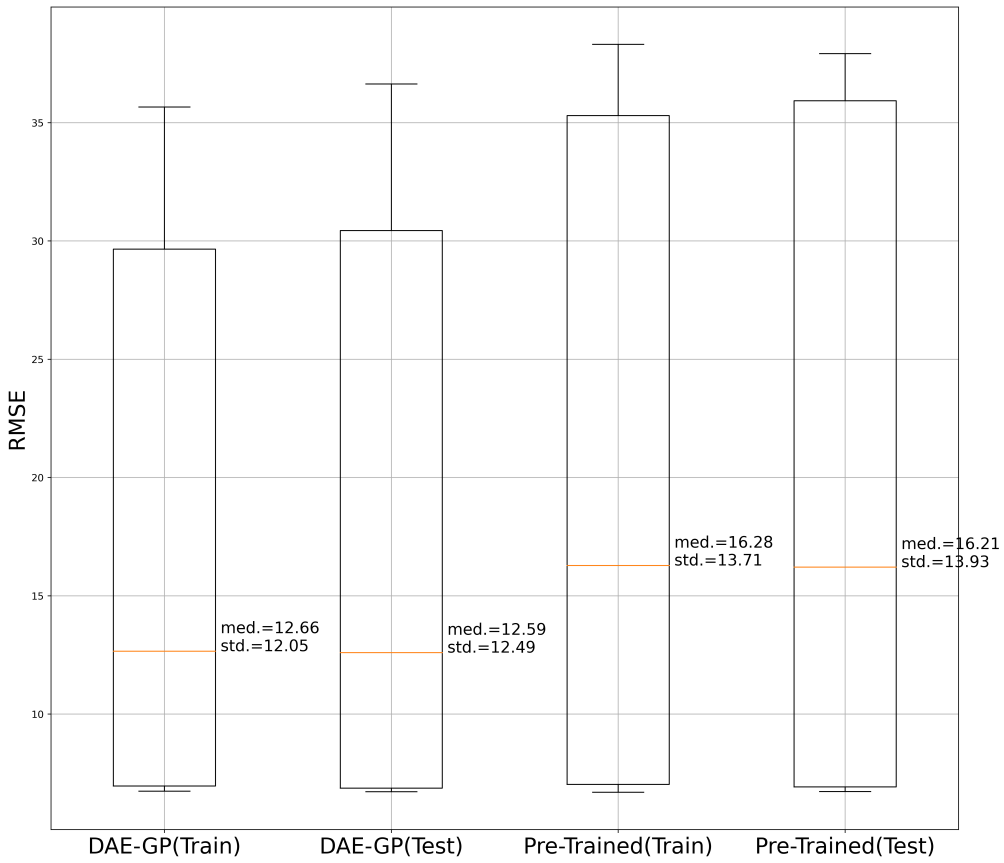

Figure 12: Fitness after 30 Generations - Airfoil - Single Hidden Layer

4.2.3 Alternative Pre-Training Strategies

Since pre-training thus far did not demonstrate significant benefits for the quality of solutions found by DAE-GP, this section describes two alternative pre-training strategies, Second Generation Pre-Training and Grow-Initialized Pre-Training, that are also tested for the airfoil dataset.

One common observation in GP-based optimization algorithms is that the largest improvements in fitness are often occurring in the first generation of the search process. Second Generation Pre-Training tries to exploit this observation by shifting the pre-training phase from the start of the algorithm into the second generation with the aim to produce a pre-training Population $\hat{P}$ that more accurately captures the properties of the search problem. The basic idea is to run regular DAE-GP for the first generation without any interference, after completing the first generation, the pre-training population $\hat{P}$ is created by sampling the DAE-LSTM model M_1. The pre-training model $\hat{M}$ is then trained using $\hat{P}$, after finishing, DAE-GP continues running and initializes each new DAE-LSTM model M_g for $g \in \{x \mid x \text{ is a number and } 2 \leq x \leq g_{max}\}$ using the trainable parameters $\theta_{\hat{M}}$.

The second alternative pre-training strategy is based on the choice of an alternative initialization algorithm to create $\hat{P}$. The pre-training implementation in previous experiments used `ramped half and half` initialization [15] to create $\hat{P}$. Using `ramped half and half`

leads to 50% of the population being initialized by the `full` method, where terminals are only inserted at the maximum allowed depth of a tree resulting in full trees. An alternative method of initializing $\hat{P}$ is to use the `grow` initialization method where solutions are not guaranteed to be full trees since every leaf below the maximum depth of a tree can also be a terminal expression. This different structural composition of solutions inside the pre-training population might be another way to improve the pre-training strategy by training on a population that more closely resembles the structural properties of further generations population where a lower percentage of full grown trees is to be expected.

Figure 13 shows the median best fitness by generations for both alternative pre-training strategies as well as the results for standard DAE-GP and the standard Pre-Training implementation used in earlier experiments. Unfortunately, both alternative strategies demonstrate an inferior performance in fitness compared to both regular DAE-GP and standard pre-trained DAE-GP resulting in lower fitness.

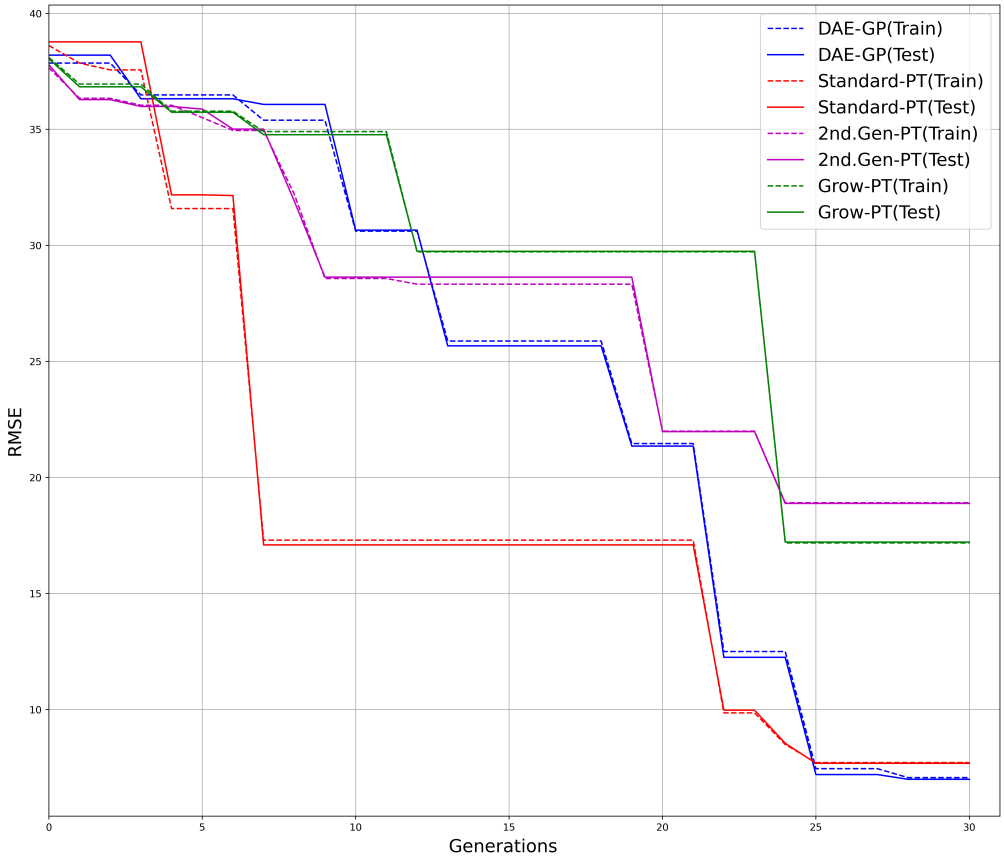

Figure 13: Best Fitness over 30 Generations - Airfoil - Alternative Pre-Training Strategies

Figure 14 shows the distribution of best fitness scores achieved on the test set after running for 30 full generations. Both alternatives, second generation Pre-Training as well as grow-initialized Pre-Training, show a much higher median RMSE (18.88/17.21) compared to standard pre-training (8.55) and regular DAE-GP (6.99). Also both approaches do not

demonstrate any benefit on the dispersion of solutions found resulting in similar standard deviations if compared to standard pre-training.

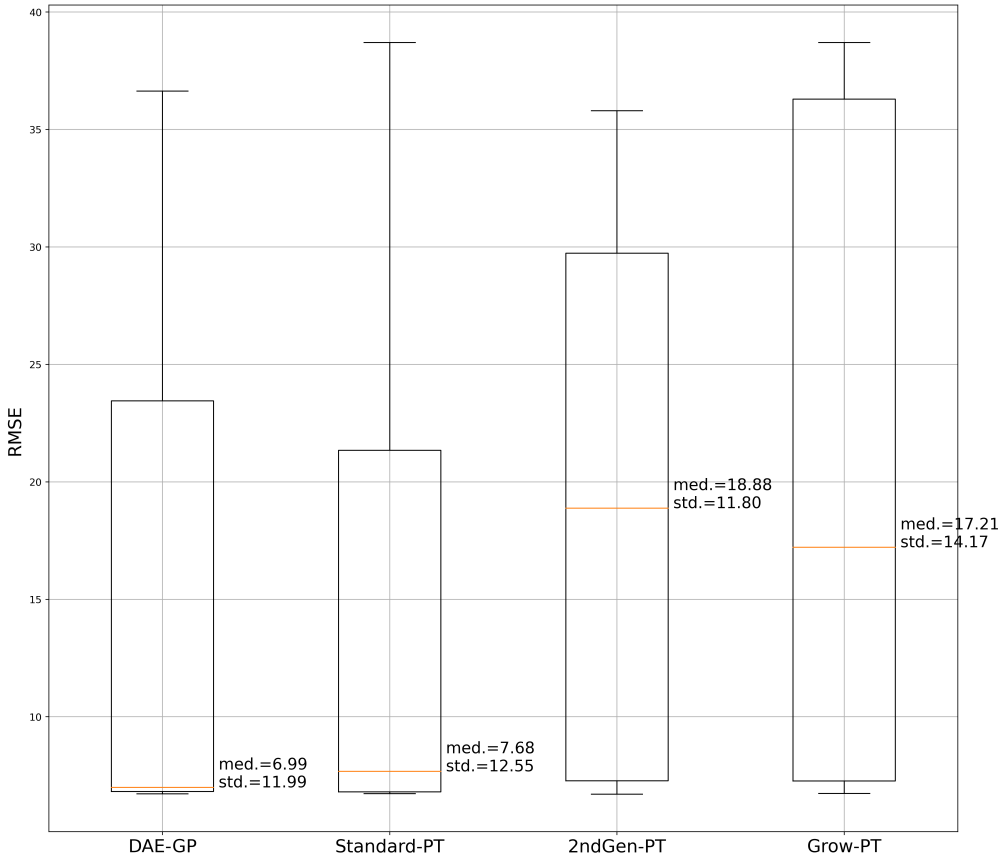

Figure 14: Best Fitness on Test Set after 30 Generations - Airfoil - Alternative Pre-Training Strategies

4.3 Influence on Run-Time

One important argument for using pre-training with DAE-GP is that it might be a potential way to reduce the overall run-time of the algorithm by reducing the training time of each generation's DAE-LSTM model. Especially in the context of optimization problems where solutions have to be found numerous times in a time efficient manner (e.g. Vehicle Routing in the Domain of Logistic Services), one major advantage of pre-training is that a pre-trained model can be re-used to optimize many instances of an optimization problem.

The run-time analysis in this chapter is based on experiments that were all executed on an Intel® Xeon® W-2245 Processor without the use of graphical processing unit (GPU) acceleration in the training of DAE-LSTM networks. Since the framework that was used is written in the, comparatively slow, python programming language, the absolute run-time measurements are only of limited meaningfulness. Nonetheless, by analyzing the relative run-time differences between DAE-GP and its pre-trained alternative, the influence of pre-training on DAE-GP run-time can be explored.

Surprisingly, as shown in figure 15, for the experiment conducted on the airfoil dataset with two hidden layers, a strong negative impact of pre-training can be observed for the median run-time of DAE-GP. If the time spend on pre-training is included into the total run-time, the median duration for pre-trained DAE-GP over all 10 runs increases by a factor of 3.121 from a median run-time of 4712.303 seconds for regular DAE-GP to 14706.47 seconds for the pre-trained algorithm. Even when the time spend for pre-training is excluded from the total run-time, pre-training still increases the run-time of DAE-GP by a factor of 2.66 to 12536.89 seconds.

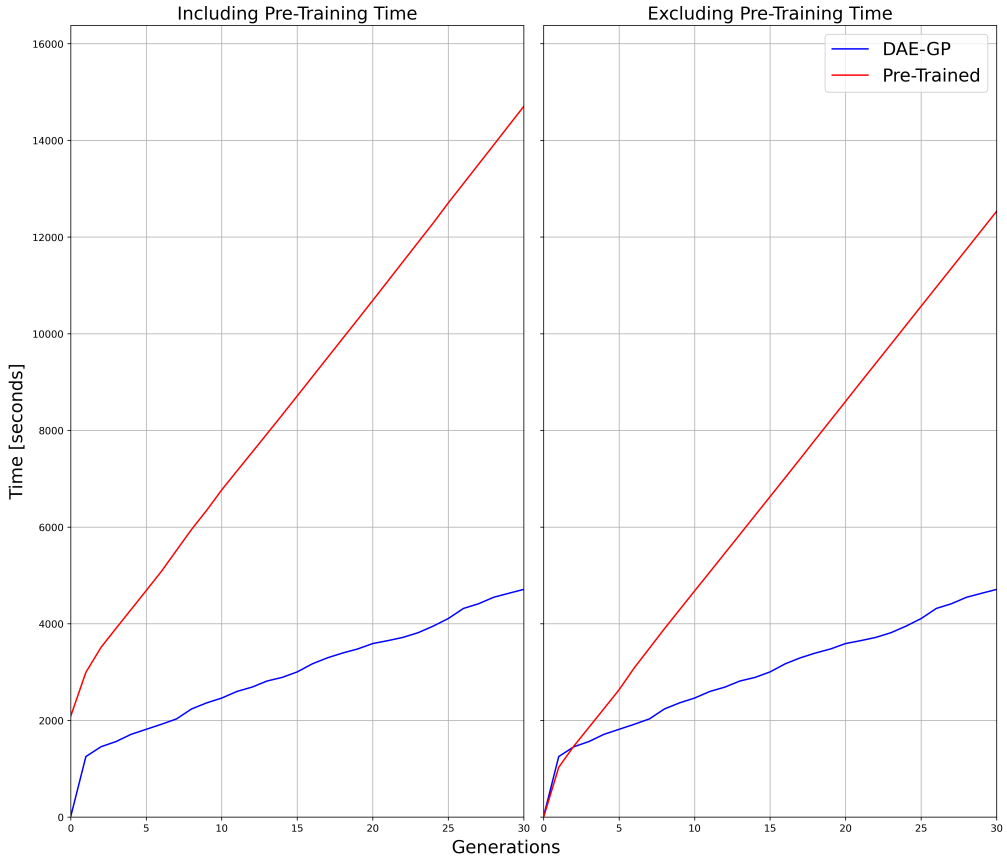

Figure 15: Median Runtime - Airfoil

Figure 16 shows the distribution of the total runtime for both DAE-GP variants. It shows that Pre-Training, besides largely increasing the median runtime, also leads to a higher dispersion which is shown by both a larger interquartile range as well as by a a standard deviation that increases by a factor of 1.232 from 635,04 for regular DAE-GP to 782,68 for pre-trained DAE-GP.

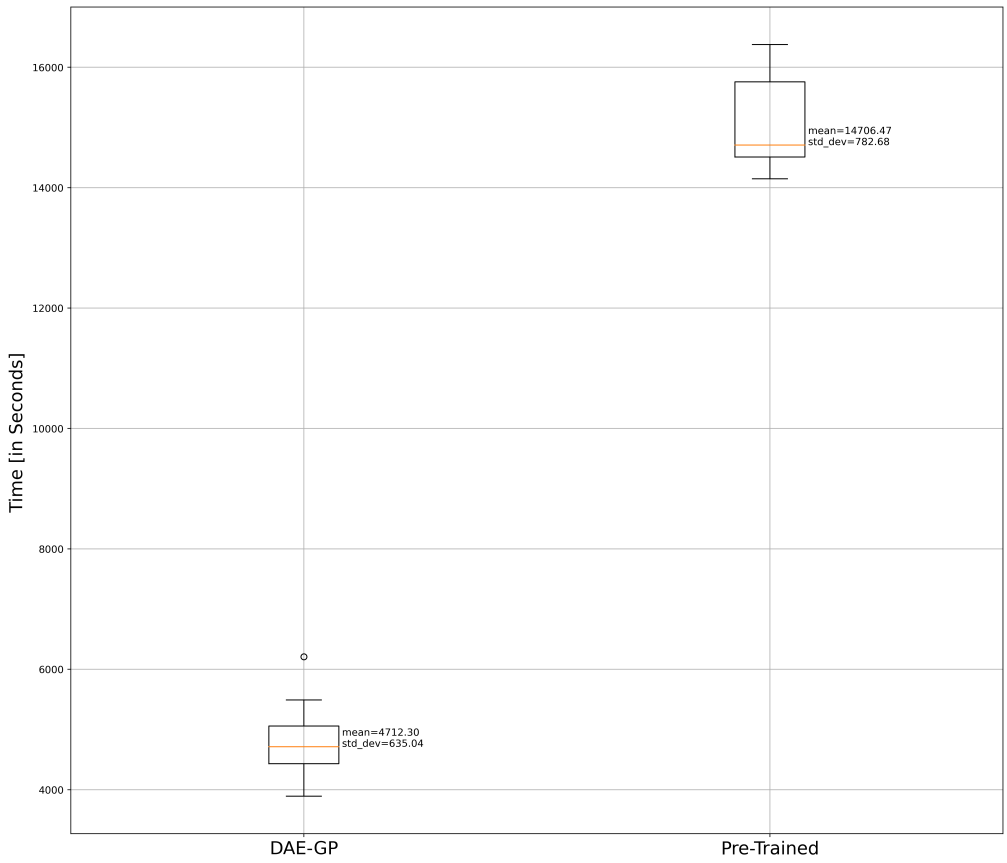

Figure 16: Total Runtime Boxplot - Airfoil

To investigate the strong negative impact of pre-training on DAE-GP run-time, the influence of pre-training on the median number of training epochs that had to be spend at each generation for the training of the DAE-LSTM models M_g (excluding the pre-training DAE-LSTM model $\hat{M}$) was studied. Figure 17 shows that pre-training, as expected, strongly reduces the number of training epochs that had to be executed at each generation until the termination condition was satisfied. While regular DAE-GP trained for a median of 122 epochs at each generation, pre-training reduces the number of median training epochs by a factor of 0.439 to 53.5 training epochs.

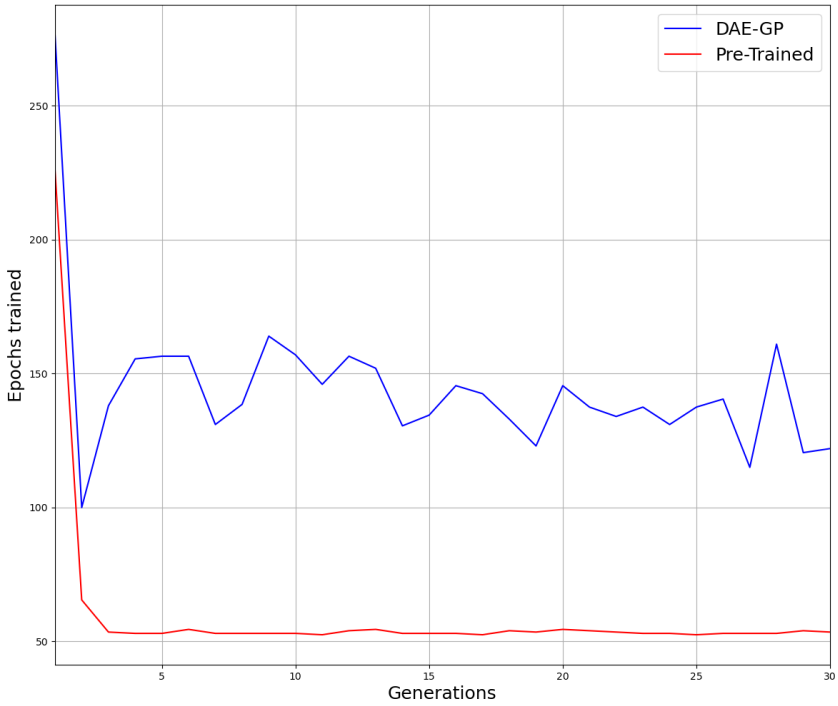

Figure 17: Median Number of Training Epochs per Generation - Airfoil

If we look at another time consuming operation in DAE-GP, the creation of new individuals by sampling the DAE-LSTM model M_g, we see one main reason for the large run-time increase by pre-training DAE-GP. Figure 18 shows the median sampling time spend during each generation of the search and it clearly demonstrates that pre-training does strongly increase the time spent on model sampling in comparison to regular DAE-GP. The median time spend on sampling new individual for regular DAE-GP increases by a factor of 6.83 from 29.05581 seconds to 198.4503 seconds.

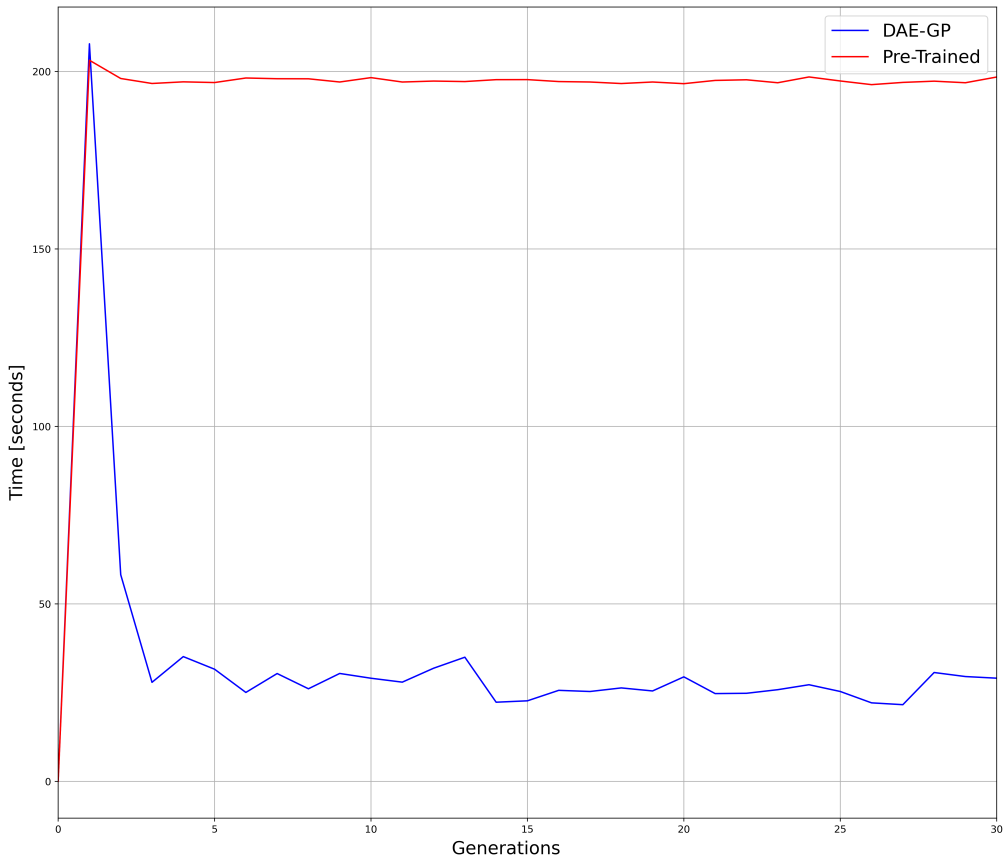

Figure 18: Median Sampling Time per Generation - Airfoil

Surprisingly, even after correcting the run-time for both DAE-GP results by removing the time spend during sampling, my implementation of pre-training for DAE-GP still on average takes more time to finish the search process as shown in figure 19. While traditional DAE-GP has a median corrected run-time of 3550.277 seconds , the corrected run-time of pre-trained DAE-GP still increases by a factor of 1.865 to 6622.027 seconds.

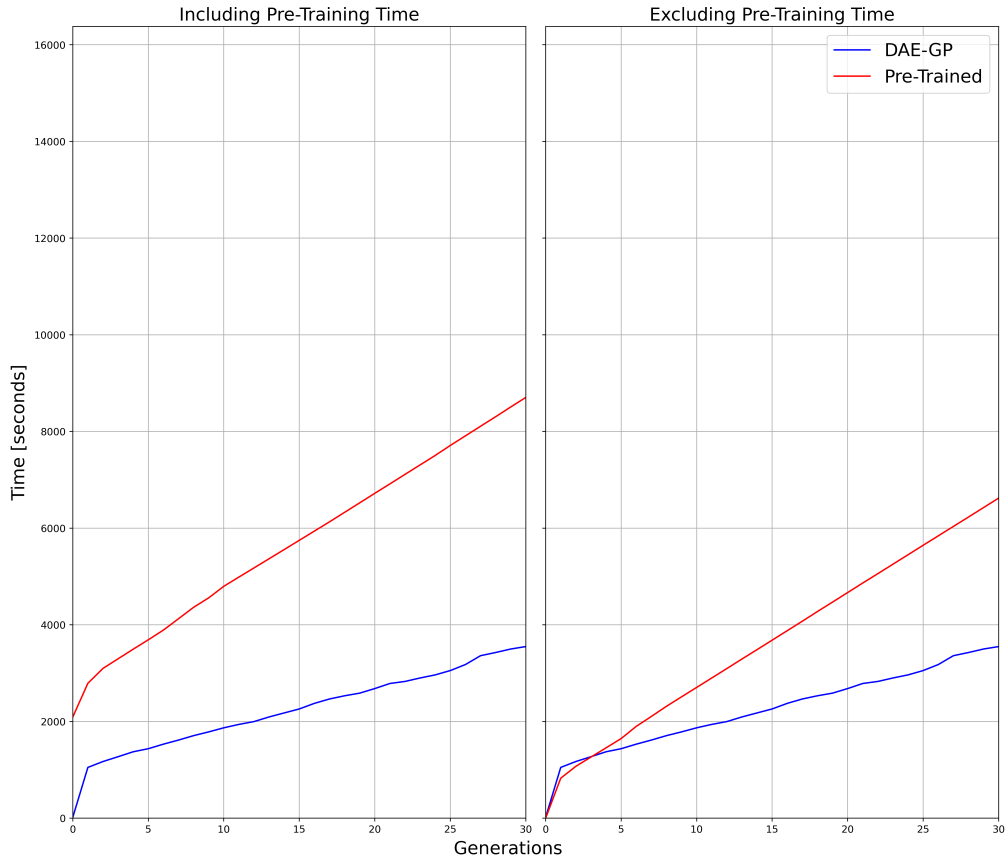

Figure 19: Median Runtime excluding Time for Sampling - Airfoil

While investigating other reasons for the differences in run-time, it was verified that both DAE-GP variants use the exact same number of trainable parameters for each generation's DAE-LSTM model M_g (total number of trainable parameters DAE-GP=573473, Pre-trained DAE-GP=573473). After profiling a single run of DAE-GP and pre-trained DAE-GP using the python `cProfiler` module, my best explanation for the remaining differences in run-time is an inefficiency in my implementation of pre-training. Figure 20 shows the cumulative time spend for the 20 most expensive function calls. Even though it is difficult to identify the exact reasons for the performance hit for pre-trained DAE-GP, it shows two interesting findings: Firstly, both algorithms spend relatively more time during model sampling (calls to `DAE_LSTM.py:208(sample)`) than during model training (calls to `training.py:1303(fit)`). Secondly, it shows that internal error handling of the baseline neural network library `keras` (calls to `traceback_utils.py:138(error_handler)` and `traceback_utils.py:59(error_handler)`) takes up a comparatively larger amount of run time for pre-trained DAE-GP than for regular DAE-GP.

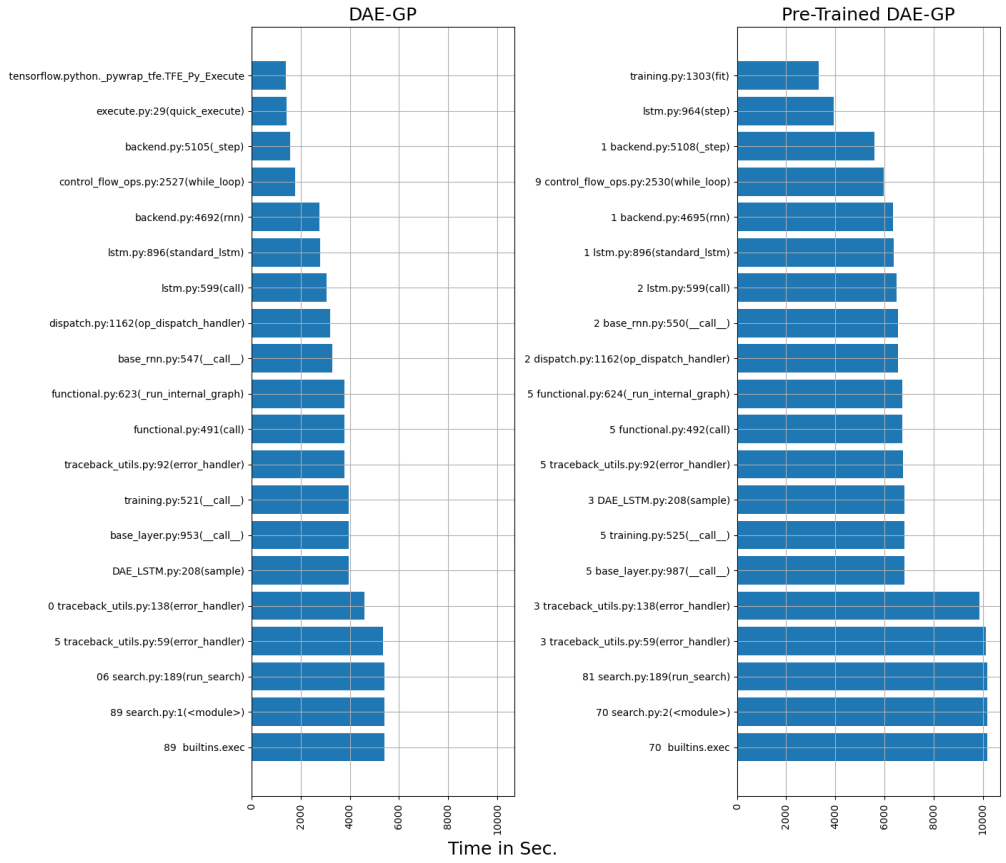

Figure 20: Cumulative Time consumption by Function Calls (Top 20) - Airfoil

5 Results over different Real-World Symbolic Regression Problems

To gather more confidence in the previous results, the pre-training strategy for DAE-GP was applied to three additional real world symbolic regression problems. Table 4 briefly summarizes the benchmark problems.

Table 4: Real World Symbolic Regression Benchmark Problems

Problem	Observations	Features	Source
Airfoil	1503	5	[1]
Boston_Housing	506	13	[9]
Energy_Cooling	768	8	[26]
Concrete	1030	8	[36]

To test for statistical differences of the results achieved by either pre-trained or regular DAE-GP, the nonparametric Mann-Whitney-U Test ([32], [17]) was selected to test for

differences in the underlying distribution. Additionally, the effect strength of pre-training as measured by Cliffs Delta [3] was included into the statistical analysis. Better median values (e.g. higher fitness) are printed bold and p-values are marked with asterisk (1,2,3) for α levels (0.1, 0.05, 0.01). Note that the results for the airfoil problem with a single hidden layer were also included into the statistical evaluation which is not visualized during this chapter (see subsection Reduced Number of Hidden Layers).

Also note that the results for the real world symbolic regression problems inside this chapter, except for the airfoil problem (see subsection Influence on Run-Time), have not been analyzed for their total run-time. This is due to the fact, that experiments were run on different computer architectures which would result in biased results.

5.1 Influence on Fitness

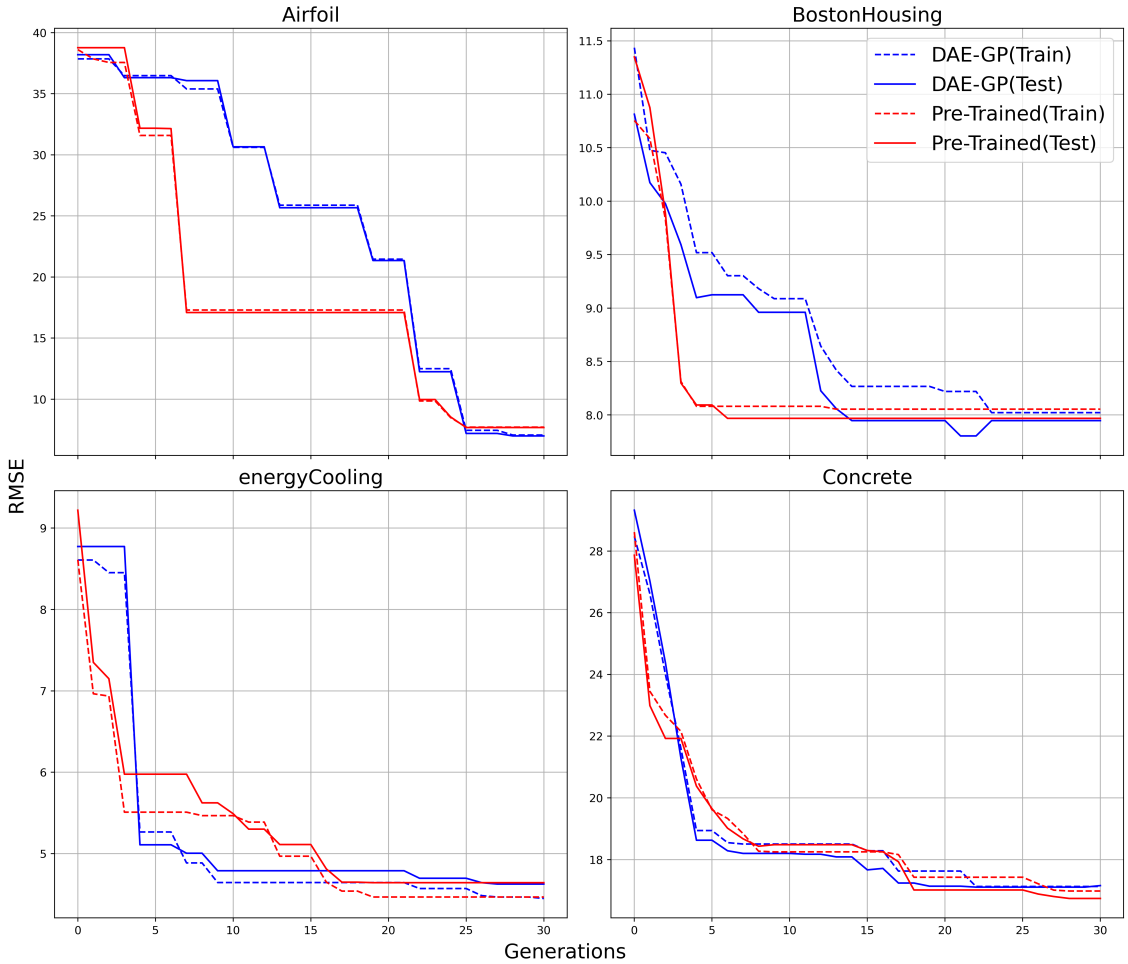

Figure 21: Fitness over 30 Generations - Real World Symbolic Regression

The results for the median best fitness over the evolutionary search are visualized in figure 21. In general, pre-training and regular DAE-GP both are able to improve on the best fitness found over the evolutionary search. Looking at figure 21, it appears that no significant

differences for the best fitness between both algorithms exist. Regular DAE-GP achieves a slightly lower RMSE on the test set for Airfoil, Boston Housing and the Energy Cooling dataset while pre-trained DAE-GP is slightly better for the Concrete dataset.

Table 5: Median Best Fitness after 30 generations - Real World Symbolic Regression

Problem	Hidden-Layers	Set	DAE-GP	Pre-Trained	P-Value	Cliffs-Delta
Airfoil	1	Train	**12.6611**	16.2778	0.57	0.16
	1	Test	**12.5944**	16.2131	0.62	0.14
Airfoil	2	Train	**7.0644**	7.7109	0.62	0.14
	2	Test	**6.991**	7.6799	0.91	0.04
Boston_Housing	2	Train	**8.0217**	8.0543	0.85	0.06
	2	Test	**7.9472**	7.9683	0.79	0.08
Energy(Cooling)	2	Train	**4.449**	4.4668	0.54	0.17
	2	Test	**4.6258**	4.6429	0.76	0.09
Concrete	2	Train	17.1299	**16.9825**	0.91	-0.04
	2	Test	17.1585	**16.7413**	0.62	-0.14

Table 5 summarizes the results for the median best fitness achieved after 30 generations and shows the results of computing the statistical tests.

As expected after reviewing figure 21, none of the differences in median best fitness are statistically significant and pre-training only has a small, negligible effect strength with Cliff's Delta ranging between -0.14 and 0.17 [28].

Figure 22 shows the distribution of final fitness values on both the test and the training set as box plots. It shows that depending on the symbolic regression problem, the dispersion of fitness values is different for both DAE-GP variants. For example, a comparatively large span is visible for the airfoil problem while fitness scores for Energy Cooling are much more concentrated. The distribution are nonetheless similar for both algorithms which does not suggest any meaningful differences between using pre-training and the distribution of final fitness scores.

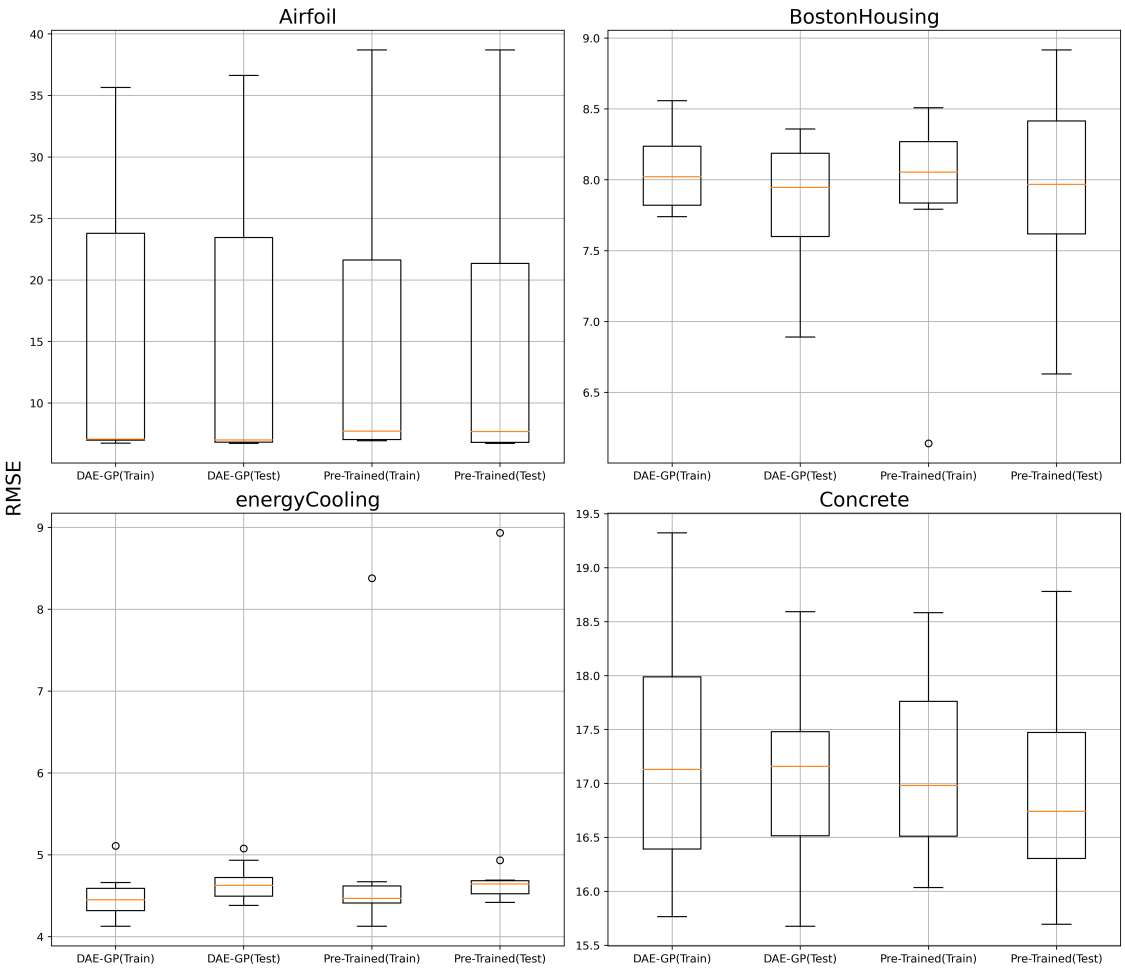

Figure 22: Fitness after 30 Generations - Real World Symbolic Regression

5.2 Influence on Solution Size

Next, the size of solutions found during DAE-GP was analyzed. Figure 23 shows the median size of the best solution per generation aggregated over all benchmark problems for symbolic regression. Similar to the previous results regarding the fitness of the best found solution, no significant differences in the size of the best found solution are visible between both DAE-GP variants.

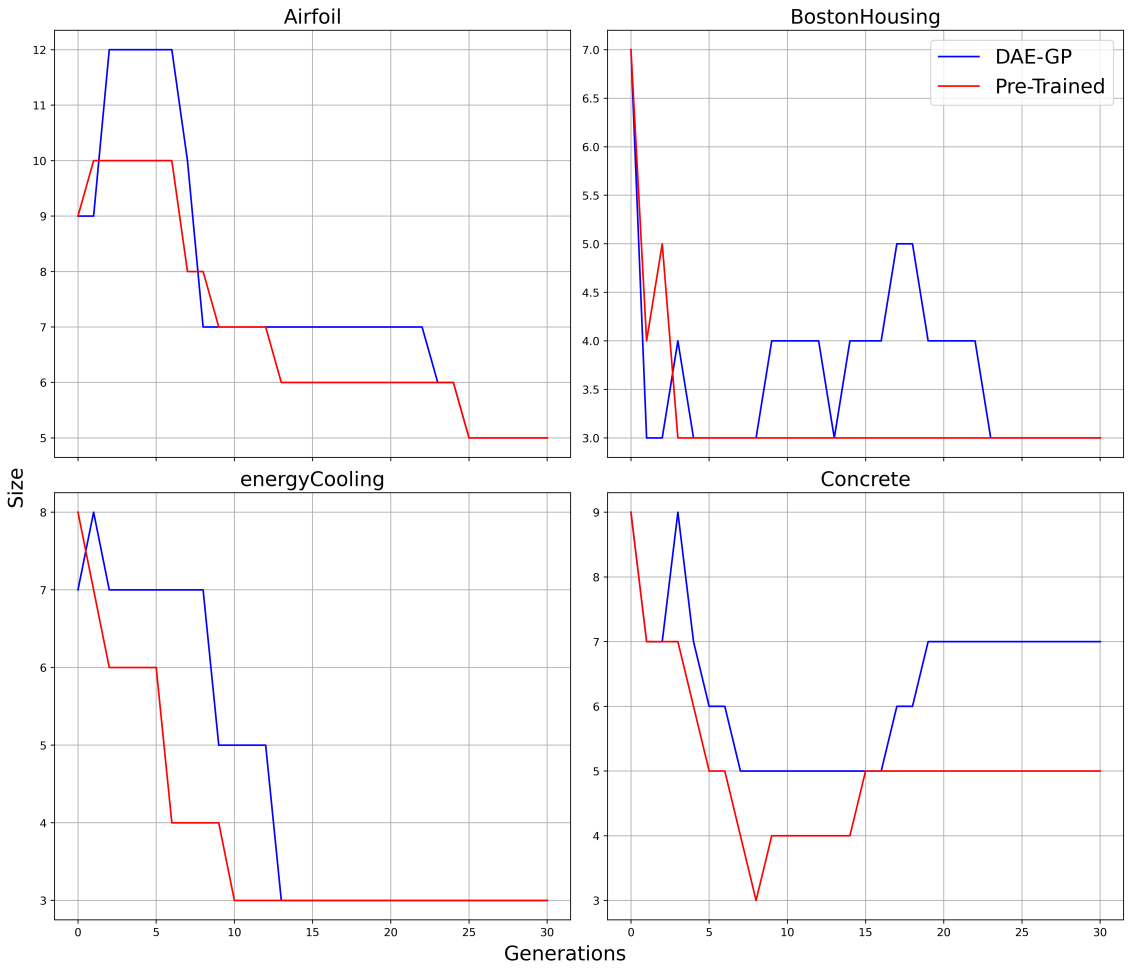

Figure 23: Size of the best Solution over 30 Generations - Real World Symbolic Regression

Table 6: Median Size of the best solution after 30 generations - Real World Symbolic Regression

Problem	Hidden Layers	DAE-GP	Pre-Trained	P-Value	Cliffs-Delta
Airfoil	1	**5**	7	0.44	0.20
Airfoil	2	5	5	0.97	-0.02
Boston_Housing	2	3	3	0.37	-0.20
Energy(Cooling)	2	3	3	0.21	-0.28
Concrete	2	7	**5**	0.21	-0.33

Table 6 shows the median size of the best solution after 30 generations and results for computing the statistical tests. For Airfoil, Boston Housing and Energy Cooling data-sets, the median size of the best solution are equal, only for the concrete dataset a smaller median size for pre-trained DAE-GP was achieved (pre-trained DAE-GP also showed a higher median

best fitness for the concrete problem). The results of the Mann-Whitney-U Test show, that differences in the underlying distributions are not statistically significant, which also shows in low effect strengths of pre-training on the size of the best found solutions (Cliff's Delta ranges from-0.33 to 0.2).

5.3 Influence on Population Diversity

Figure 24 shows the median levenshtein edit distance as a metric for population diversity over the evolutionary search. It clearly shows that pre-training does increase the population diversity for DAE-GP consistently for all tested benchmark problems.

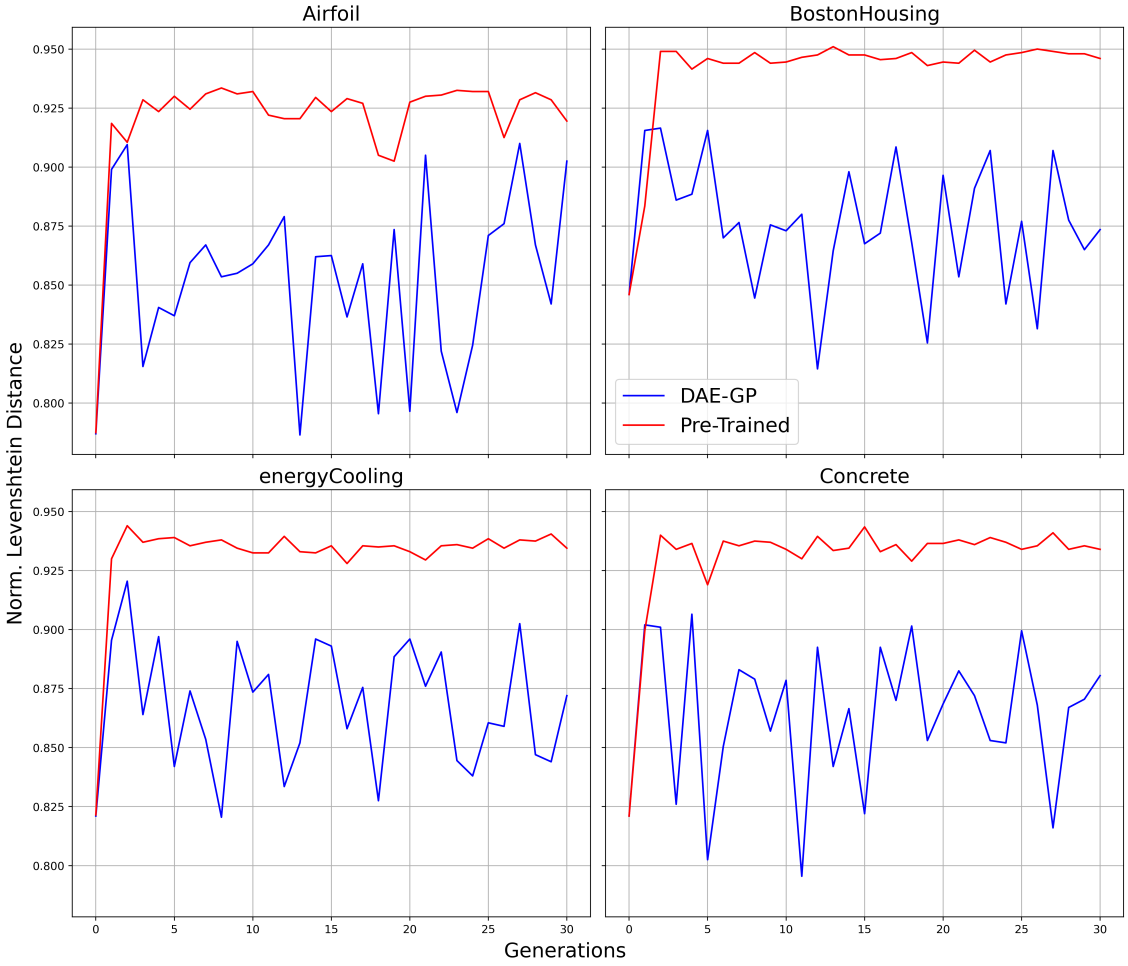

Figure 24: Population Diversity over 30 Generations - Real World Symbolic Regression

Table 7: Median Population Diversity over 30 Generations - Symbolic Regression

Problem	Hidden Layers	DAE-GP	Pre-Trained	P-Value	Cliffs-Delta
Airfoil	1	0.86	**0.93**	0.00***	0.93
Airfoil	2	0.86	**0.93**	0.00***	0.93
Boston_Housing	2	0.88	**0.95**	0.00***	0.92
Energy(Cooling)	2	0.87	**0.94**	0.00***	0.94
Concrete	2	0.87	**0.94**	0.00***	0.93

The results of the statistical analysis are summarized in table 7. For all sub experiments, computing the Mann-Whitney-U Test returns P-Values smaller than 0.01 which indicates a highly statistically significant difference in the underlying distributions. Looking at the effect strength of pre-training, it shows that pre-training does have a strong positive effect (Cliff's Delta ranges from 0.92 to 0.94) on population diversity inside the population. Pre-Training did increase the median normalized levenshtein distance over all generations and sub-experiments by a factor of 1.077 from 0.869 to 0.935.

5.4 Influence on the number of Training epochs per Generation

Finally, figure 25 shows the median number of epochs spend on training each generations DAE-LSTM network M_g until the termination criterion was satisfied. Pre-Training shows to be very effective at reducing the number of training epochs in comparison to regular DAE-GP.

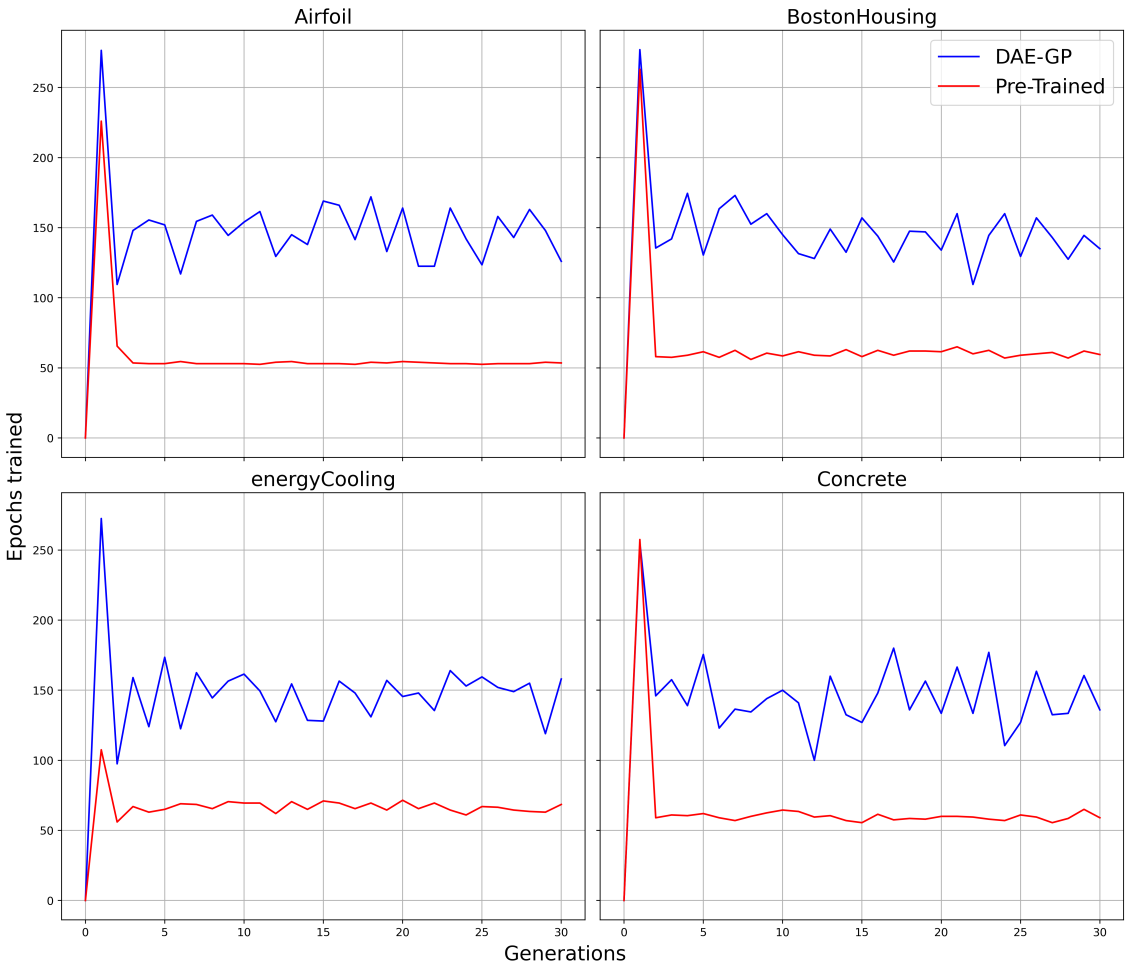

Figure 25: Training Epochs over 30 Generations - Real World Symbolic Regression

Table 8: Median Number of Training Epochs per Generation - Symbolic Regression

Problem	Hidden Layers	DAE-GP	Pre-Trained	P-Value	Cliffs-Delta
Airfoil	1	143.49	**64.66**	0.00***	-0.88
Airfoil	2	143.83	**61.22**	0.00***	-0.88
Boston_Housing	2	141.8	**65.36**	0.00***	-0.87
Energy(Cooling)	2	144.24	**66.7**	0.00***	-0.93
Concrete	2	139.2	**65.15**	0.00***	-0.87

Looking at the results of the statistical analysis summarized in table 8, computing the Mann-Whitney-U test does consistently return P-Values below 0.01 for all experiments. Analyzing the effect strength of pre-training on the number of training epochs shows values for Cliff's Delta in the range of -0.93 and -0.87 which indicates a strong negative effect strength. This

evidence suggests that pre-training can substantially decrease the required number of training epochs, indicating a statistically significant impact.

Computing the median number of training epochs over all generations for all benchmark problems shows a total decrease by a factor of 0.45 from 143.49 epochs for regular DAE-GP to 65.15 epochs for pre-trained DAE-GP.

6 Conclusion and Discussion

6.1 Benefits of using Pre-Training

The experiments conducted in the scope of this master thesis did find that the implemented pre-training strategy resulted in a highly statistically significant increase in population diversity measured by the normalized levenshtein edit distance (see subsection Influence on Population Diversity). Increasing the population diversity can be an important mechanism in controlling the search behavior of evolutionary algorithms and is especially important for the global exploration of the search space and the avoidance of premature convergence of the search [24].

Another significant benefit of using pre-training in DAE-GP is that it is highly efficient in decreasing the number of training epochs that are necessary to train each generation´s DAE-LSTM model (see subsection Influence on Population Diversity). Even though pre-training increased the median total run-time for the tested benchmark problem in combination with the chosen DAE-LSTM dimensions, reducing the number of training epochs might still be a possible way to lower the run time of DAE-GP if further performance optimizations, especially during model sampling, can be achieved.

6.2 Disadvantages of using Pre-Training

The experiments conducted in this thesis did not find any significant benefits, in terms of solution quality or run-time, when using pre-trained DAE-GP compared to regular DAE-GP for solving real-world symbolic regression problems.

As detailed in the subsection Dynamic Adjustment of Hidden Neurons, one major drawback of using pre-training with DAE-GP is the loss of the ability to dynamically adjust the number of hidden neurons to the maximum size of individuals inside the current generations population. This dynamic mechanism does reduce the computational resources necessary for training DAE-LSTM networks which is one of the most time consuming operations besides sampling new individuals.

Another disadvantage of using pre-training in DAE-GP is that, in my experiments, it required DAE-LSTM networks to use two hidden layers (see subsection Reduced Number of Hidden Layers). For a single hidden layer, pre-training did show a negative impact on the median final fitness of solutions found as it was suggested by [4]. Since current research on DAE-GP for real-world symbolic regression relies on a single hidden layer (see [34]), pre-training does come with additional computational expenses for using deeper DAE-LSTMs.

6.3 Limitations and open Questions

One major limitation for the experiments conducted during the research phase of this thesis has been the large computational effort for experimenting with DAE-GP. Since single runs for DAE-GP have a run-time of many hours, it was not possible to extend the experiments to larger population sizes or more generations. Also no extended optimization of hyperparameters (e.g by using a grid search strategy) have been performed which might yield configurations that increase the benefits of pre-training. As described in subsection Influence on Run-Time, sampling new solutions has been a large contributor to the large performance hit on run-time by using pre-training. It could be a worthwhile endevour to explore the main factors that increase sampling time for pre-trained DAE-GP to more efficiently exploit the benefits of pre-training such as the reducing the number of training epochs.

Another limitation of this research is the software implementation used for all experiments. As briefly touched in the subsection Influence on Run-Time, some evidence was found that my implementation of pre-training suffered from increased run-time costs for internal error handling in the used `keras` library for deep neural networks. By optimizing the source code or even re-implementing it in a more performance-focused programming language (such as `C++` or `Rust`), large improvements to the pre-training strategy might be achievable.

Another possibility to improve the benefits of pre-training for DAE-GP is based on the findings of [5]. Pre-Training has shown to be especially useful for the lower layers of DAEs therefore it could be worthwhile to further experiment with only adapting the weights for a subset of the hidden layers inside the DAE-LSTM networks to either reduce run-time or to improve on the quality of solutions found during the evolutionary search.

Nomenclature

Symbols

Table 9: List of Mathematical Symbols

Symbol	Meaning
g_n	n.th generation
g_{max}	Maximum number of generations
M_g	DAE-LSTM model for generation g
$\hat{M}$	DAE-LSTM model for Pre-Training
θ_M	Trainable parameters of DAE-LSTM model M
P_g	Population of generation g
$\hat{P}$	Pre-Training Population
$\hat{P}_{train}$	Pre-Training Population used for training DAE-LSTM model $\hat{M}$
$\hat{P}_{test}$	Pre-Training Population used for validating DAE-LSTM model $\hat{M}$
X_g	Selected Training Population for DAE-LSTM model M_g in generation g

Abbreviations

Table 10: List of Abbreviations

Abbreviation	Meaning
DAE-GP	Denoising Autoencoder Genetic Programming
GP	Genetic Programming
EDA	Estimation of Distribution Algorithm
EC	Evolutionary Computation
DAE	Denoising Autoencoder
DAE-LSTM	Denoising Autoencoder Long Short Term Memory Network
RMSE	Root Mean Squared Error
AQ	Analytic Quotient

References

[1] Thomas F. Brooks, Dennis S. Pope, and Michael A. Marcolini. 1989. Airfoil self-noise and prediction.

[2] Francois Chollet and others. 2015. Keras. Retrieved from https://github.com/fchollet/keras

[3] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114, (1993), 494–509.

[4] Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. 2010. Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (Proceedings of machine learning research), PMLR, Chia Laguna Resort, Sardinia, Italy, 201–208. Retrieved from https://proceedings.mlr.press/v9/erhan10a.html

[5] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. 2009. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Proceedings of the twelth international conference on artificial intelligence and statistics* (Proceedings of machine learning research), PMLR, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 153–160. Retrieved from https://proceedings.mlr.press/v5/erhan09a.html

[6] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research* 13, (July 2012), 2171–2175.

[7] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji-Rong Wen, Jinhui Yuan, Wayne Xin Zhao, and Jun Zhu. 2021. Pre-trained models: Past, present and future. *AI Open* 2, (2021), 225–250. DOI:https://doi.org/https://doi.org/10.1016/j.aiopen.2021.08.002

[8] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (September 2020), 357–362. DOI:https://doi.org/10.1038/s41586-020-2649-2

[9] David Harrison and Daniel Rubinfeld. 1978. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management* 5, (March 1978), 81–102. DOI:https://doi.org/10.1016/0095-0696(78)90006-2

[10] G. E. Hinton and R. R. Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science* 313, 5786 (2006), 504–507. DOI:https://doi.org/10.1126/science.1127647

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, (December 1997), 1735–80. DOI:https://doi.org/10.1162/neco.1997.9.8.1735

[12] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. DOI:https://doi.org/10.1109/MCSE.2007.55

[13] Kangil Kim, Yin Shan, Nguyen Hoai, and Robert McKay. 2014. Probabilistic model building in genetic programming: A critical review. *Genetic Programming and Evolvable Machines* 15, (June 2014). DOI:https://doi.org/10.1007/s10710-013-9205-x

[14] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980, (2014).

[15] John R. Koza. 1993. Genetic programming - on the programming of computers by means of natural selection. In *Complex adaptive systems*.

[16] Joseph B. Kruskal. 1983. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM Rev.* 25, 2 (April 1983), 201–237. DOI:https://doi.org/10.1137/1025045

[17] Henry B. Mann and Douglas R. Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics* 18, (1947), 50–60.

[18] Ji Ni, Russ H. Drieberg, and Peter I. Rockett. 2013. The use of an analytic quotient operator in genetic programming. *IEEE Transactions on Evolutionary Computation* 17, 1 (February 2013), 146–152. DOI:https://doi.org/doi:10.1109/TEVC.2012.2195319

[19] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (2010), 1345–1359. DOI:https://doi.org/10.1109/TKDE.2009.191

[20] Grégory Paris, Denis Robilliard, and Cyril Fonlupt. 2004. Exploring overfitting in genetic programming. In *Artificial evolution*, Springer Berlin Heidelberg, Berlin, Heidelberg, 267–277.

[21] Malte Probst and Franz Rothlauf. 2020. Harmless overfitting: Using denoising autoencoders in estimation of distribution algorithms. *Journal of Machine Learning Research* 21, 78 (2020), 1–31. Retrieved from http://jmlr.org/papers/v21/16-543.html

[22] Franz Rothlauf. 2011. *Design of modern heuristics: Principles and application*. DOI:https://doi.org/10.1007/978-3-540-72962-4

[23] Dirk Schweim, David Wittenberg, and Franz Rothlauf. 2021. On sampling error in genetic programming. *Natural computing* 2021, (2021). DOI:https://doi.org/http://doi.org/10.25358/openscience-5820

[24] Dirk Sudholt. 2020. The benefits of population diversity in evolutionary algorithms: A survey of rigorous runtime analyses. In *Theory of evolutionary computation: Recent developments in discrete optimization*, Benjamin Doerr and Frank Neumann (eds.). Springer International Publishing, Cham, 359–404. DOI:https://doi.org/10.1007/978-3-030-29414-4_8

[25] The pandas development team. 2020. Pandas-dev/pandas: pandas. DOI:https://doi.org/10.5281/zenodo.3509134

[26] Athanasios Tsanas and Angeliki Xifara. 2012. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings* 49, (June 2012), 560–567. DOI:https://doi.org/10.1016/j.enbuild.2012.03.003

[27] Guido Van Rossum and Fred L. Drake. 2009. *Python 3 reference manual*. CreateSpace, Scotts Valley, CA.

[28] András Vargha and Harold D Delaney. 2000. A critique and improvement of the CL common language effect size statistics of McGraw and wong. *Journal of Educational and Behavioral Statistics* 25, (2000), 101–132.

[29] Pascal Vincent, Hugo Larochelle, Y. Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on machine learning* (ICML '08), Association for Computing Machinery, New York, NY, USA, 1096–1103. DOI:https://doi.org/10.1145/1390156.1390294

[30] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17, (2020), 261–272. DOI:https://doi.org/10.1038/s41592-019-0686-2

[31] Andreas Weigend. 1994. On overfitting and the effective number of hidden units. In *Proceedings of the 1993 connectionist models summer school*, 335–342.

[32] Frank. Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics* 1, (1945), 196–202.

[33] David Wittenberg. 2022. Using denoising autoencoder genetic programming to control exploration and exploitation in search. In *Genetic programming*, Springer International Publishing, Cham, 102–117.

[34] David Wittenberg and Franz Rothlauf. 2022. Denoising autoencoder genetic programming for real-world symbolic regression. In *Proceedings of the genetic and evolutionary computation conference companion* (GECCO '22), Association for Computing Machinery, New York, NY, USA, 612–614. DOI:https://doi.org/10.1145/3520304.3528921

[35] David Wittenberg, Franz Rothlauf, and Dirk Schweim. 2020. DAE-GP: Denoising autoencoder LSTM networks as probabilistic models in estimation of distribution genetic programming. In *Proceedings of the 2020 genetic and evolutionary computation conference* (GECCO '20), Association for Computing Machinery, New York, NY, USA, 1037–1045. DOI:https://doi.org/10.1145/3377930.3390180

[36] I-Cheng Yeh. 1998. Modeling of strength of high-performance concrete using artificial neural networks." cement and concrete research, 28(12), 1797-1808. *Cement and Concrete Research* 28, (December 1998), 1797–1808. DOI:https://doi.org/10.1016/S0008-8846(98)00165-3

Statutory Declaration

Statutory Declaration

I hereby assure that I have independently and without using sources or resources other than those indicated, written this paper. Directly quoted sentences or sentence parts are marked as citations, while other sources of inspiration, in terms of content and scope, are identified with references. The work has not been presented to any examination board in the same or a similar form and has not been published. It has not been used, even partially, for any other examination or academic achievement. I am aware of the regulations for ensuring good scientific practice in research and teaching, as well as the procedures for dealing with scientific misconduct.

Place, Date: Mainz 16.2.2023

Signature:

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Wörtlich übernommene Sätze oder Satzteile sind als Zitat belegt, andere Anlehnungen, hinsichtlich Aussage und Umfang, unter Quellenangabe kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und ist nicht veröffentlicht. Sie wurde nicht, auch nicht auszugsweise, für eine andere Prüfungs- oder Studienleistung verwendet. Von der Ordnung zur Sicherung guter wissenschaftlicher Praxis in Forschung und Lehre und zum Verfahren zum Umgang mit wissenschaftlichem Fehlverhalten wurde Kenntnis genommen wurde.

Ort, Datum: Mainz 16.2.2023

Unterschrift: